MINT Capstone Project Report

NAME: Samil Vanzar

Project Title: Redesign of IPv4 for small (radio) control systems purposes

Project Mentor: Professor Leonard Rogers

Acknowledgement

I would like to express my deepest gratitude to Professor Leonard Rogers for his genuine guidance and helpful attitude throughout this project work. I am grateful as my honorable Supervisor devoted his valuable time and shared his expertise knowledge.

I would like to express my great appreciation to Dr. Mike Macgregor for giving me an opportunity to work on this project. I am thankful to him for sparing his valuable time from his busy schedules.

I am also thankful to Professor Shahnawaz Mir for providing me an opportunity to perform this project.

Finally, I would also like to acknowledge the help provided by my friends, colleagues and to every person who helped me for successful completion of the project, both explicitly or implicitly.

Samil Vanzar

Table of Contents

List of figures

Introduction

Security of IPv4 packets is of great importance as it carries user's important data in the payload. There are different technologies like VPN, SSL that uses different security mechanisms to protect user's data from being accessed by the attackers. However, with the advancement in technologies there are several ways available for attackers to steal user's data.

In this project, we are implementing one of the ways for protecting data from illegitimate access to the attacker by changing the header structure of the IPv4 packets. Attacker will not be able to spoof the packets as default structure of IPv4 header is changed. This will allow safe transportation of data over untrusted Public Network.

In this project, data is sent over UDP Protocol using custom IPv4 Header packets. This is implemented using two virtual systems in same broadcast domain in Virtual Box Environment using Raspbian OS. Below changes are made to default IPv4 header structure:

Ethertype is selected as 0xBAD0 instead of 0x0800, for indicating OS that frame is carrying IPv4 data. Ethertype was changed to 0xBAD0 for protecting data from getting spoofed at Data Link layer.

Updated IPv4 header will use below values:

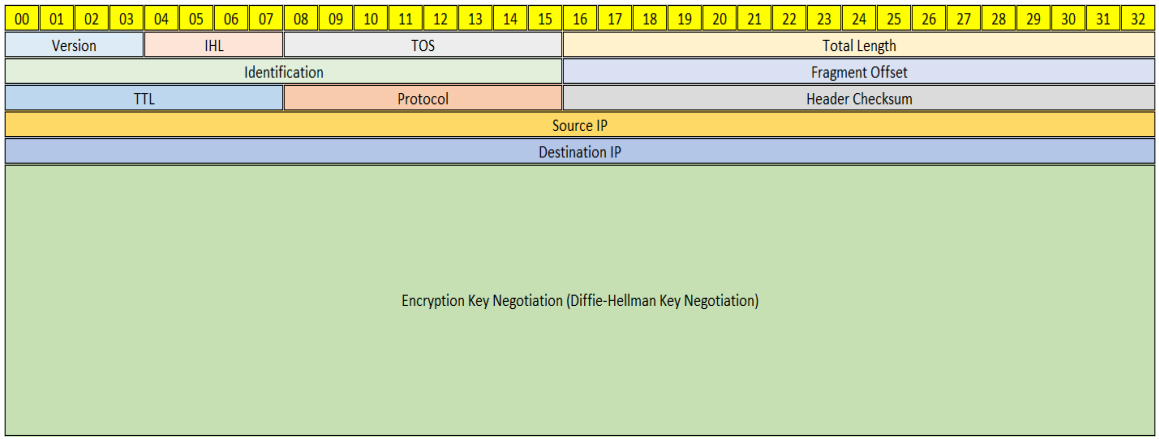


Figure 1: Custom IPv4 Header Structure

Version Field = 1. This field will separate custom IPv4 packets from default IP packets.

IHL = 15 (always) for making custom IPv4 packets compatible to default IPv4 packets and indicating that option field is always present and carried in the header.

TOS = Using custom Type of service we would be able to send different commands to the control systems like mode of operation, if operation is critical or if it requires encryption. Different TOS options are,

Normal Operation = 0x00;

Immediate Operational Shutdown = 0x01;

Routine Operation Response = 0x02;

Priority Operation Response = 0x03;

Critical Operation Response = 0x04;

Manual Control / Emergency Take Over = 0x05;

Regular Data Reporting = 0x06;

Priority Data Reporting = 0x07;

Immediate Data Reporting = 0x08;

Streaming Data Reporting = 0x09;

Perform System Check = 0x10;

Report System Status = 0x11;

Immediate Report In – Return Home = 0x12;

Encrypted Operation – Renegotiate = 0x13;

Encrypted Operation – Normal = 0x14;

Total Length Field: Operates the same as default IPv4 header Length field.

Identification Field: Operates the same as default IPv4 Identification field.

Fragment Offset Field: Operates the same as default IPv4 Fragment Offset field.

TTL Field: Operates the same as the default IPv4 TTL field.

Protocol Field: Operates the same as default IPv4 protocol field.

Header Checksum Field: Operates the same as default IPv4 Header checksum field.

Source IP Field: Operates like default IPv4 header source field, however is parsed differently.

Destination IP Field: Operates like default IPv4 header destination field, however is parsed differently.

Encryption Key Negotiation Field: This field is introduced to secure IPv4 packets by adding encryption parameter in Option field. This will help in providing encryption to IPv4 packets

without having to use additional security mechanisms like IPsec for encrypting packet and sending them to remote network. This field is controlled by TOS field,

If TOS = 0x14, data needs to be encrypted and sent to the remote end along with DH Key.

For any other value of TOS, operation is unencrypted and the key to be sent to the remote end is:

01110011 10001100 01110011 10001100

The IP Address assignment is functionally different in this new header. The first octet defines the geographic region that the device is operating in. The second octet refers to the corporate subnet that the device is operating in, the third and fourth octets refers to the individual device itself. Objective behind changing IP address field is to protect packets from getting spoofed by hackers at Layer – 3. Only the sender and receiver control systems have updated IPv4 headers and would be able to understand the message. Data, encrypted or unencrypted, present in the custom IPv4 packet is sent to the remote end over UDP protocol using port 787.

Project Environment Setup

Logical Diagram

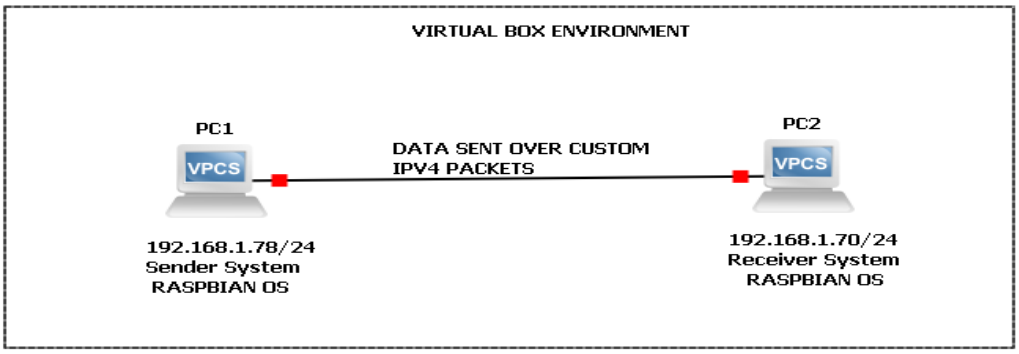


Figure 2: Project Scenario implemented in Virtual Box

Components (S/W & H/W) used in the project

As shown in logical diagram, two virtual machines with **Raspbian OS (Linux Raspberry 4.9.0-8-amd64 Debian**) were installed on Windows 7 OS machine using ***oracle virtual box (version – 5.1.28r117968)*** to send data over custom IPv4 header unidirectionally. IP address and subnet mask of the systems are as follows:

Sender System –

IP ADDRESS = 192.168.1.78 (STATIC)

SUBNET MASK = 255.255.255.0

Receiver System –

IP ADDRESS = 192.168.1.70 (STATIC)

SUBNET MASK = 255.255.255.0

Wireshark (version 2.2.6) is used as packet sniffing tool to verify that data is being sent and received over custom IPv4 header. ***Leafpad Editor (version 0.8.18.1)*** is used as editing tool to edit files or to write new files on Raspberry

Project Workflow

STEP 1: BASIC INSTALLATION AND SETUP

For implementing this project in virtual Environment, virtual image of Raspbian OS was installed in virtual box. Network settings in virtual box were set to Bridge Adapter to access internet. Once the sender device was setup, its clone was created as receiver device.

By default, the network card in virtual Raspberry OS is set to take IP address via DHCP, so that setting was changed to static IP by editing below file at sender and receiver end:

`/etc/dhcpcd.conf file`

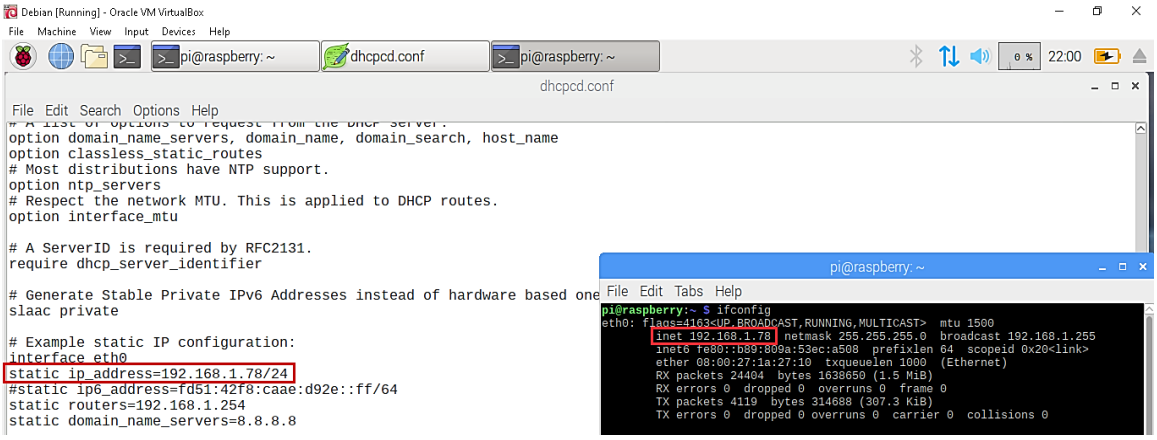


Figure 3: Static IP Configuration in Sender System

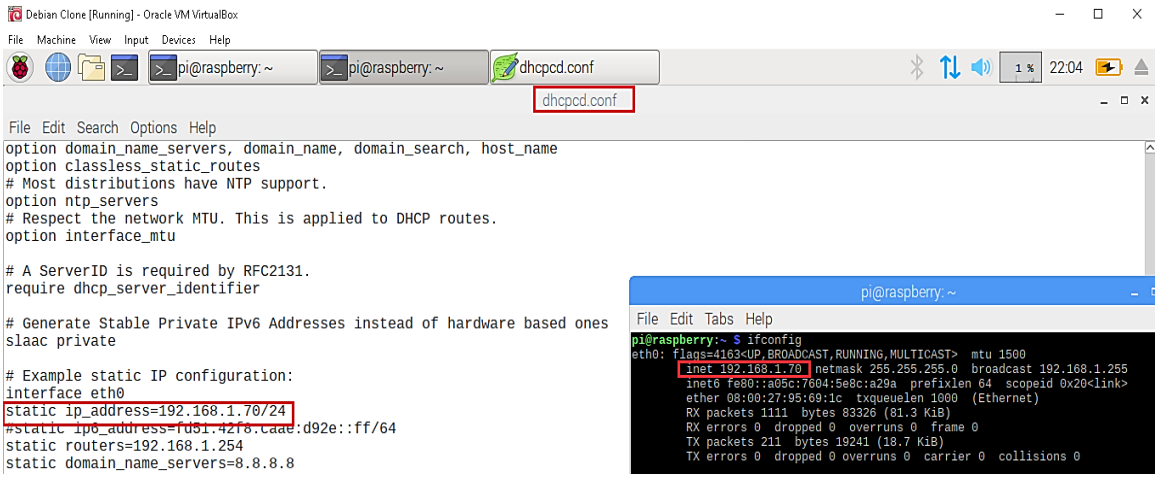


Figure 4: Static IP Configuration in Receiver System

All the required software's were installed on Raspbian OS and the connectivity between sender and receiver virtual machine was verified.

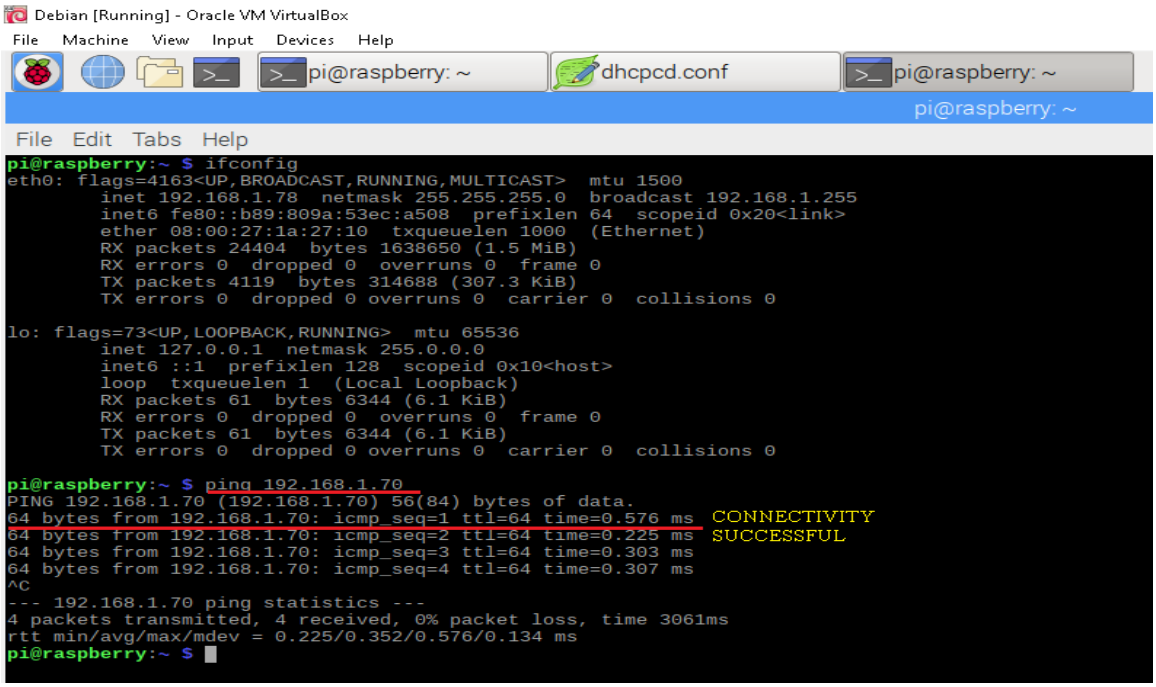


Figure 5: Connectivity verification between two systems

STEP 2: CHANGING ETHERTYPE FIELD

After successfully setting up the devices, it was required to change the Ethertype from 0x0800 to 0xBAD0. Changing Ethertype field will help protecting data as attacker system would not be able to receive it because default TCP stack is designed to understand default Ethertype of 0x0800.To achieve this, it was required that OS understands that when Ethertype 0xBAD0 is used, it means frame has IPv4 packets. This was achieved by creating custom socket. In the sender and receiver programs this part of the program is defined as:

```
sockfd = socket (AF_PACKET, SOCK_RAW, IPPROTO_RAW)
```

- AF_PACKET = **AF_PACKET** is used to play with packets at the protocol level, i.e. implementing custom protocol. It allows the implementation of protocol modules in user space on top of the physical layer.
- SOCK_RAW = The socket_type is **SOCK_RAW** for raw frames at Data Link Layer.
- IPPROTO_RAW = Used to edit the header of IP packet**.**

To test if modified Ethertype was working after socket configurations, data was required to be sent to the remote system. For sending data destination MAC address, destination IP address, and ethernet header was required to be defined.

Once the MAC address was defined in the program, data “Hello world! “was sent. The sender code mentioned in Appendix A, highlights the section defining MAC addresses of the receiver system.

Below image shows the MAC address of the receiver machine:

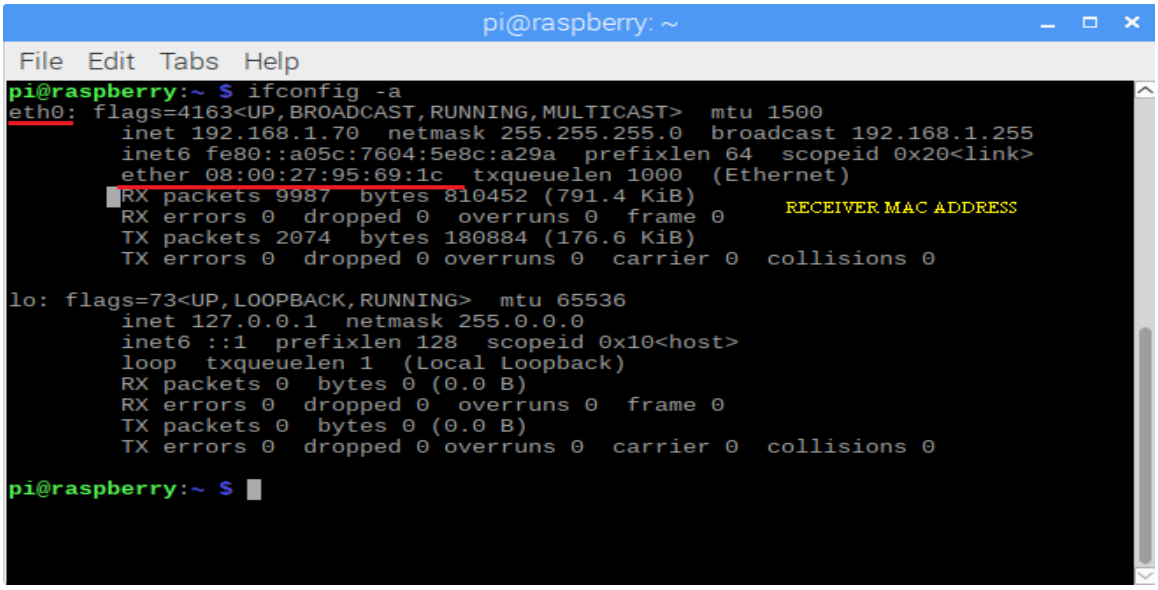


Figure 6: MAC Address of Receiver

Once the data was sent, it was necessary to verify if it was received at the receiver end using custom Ethertype 0xBAD0. To achieve this, Wireshark packet capture tool was installed and ran on both sender and receiver machines on eth0 interface and results were verified. Following syntax was used to run program:

```
sudo ./<file name given to program>
```

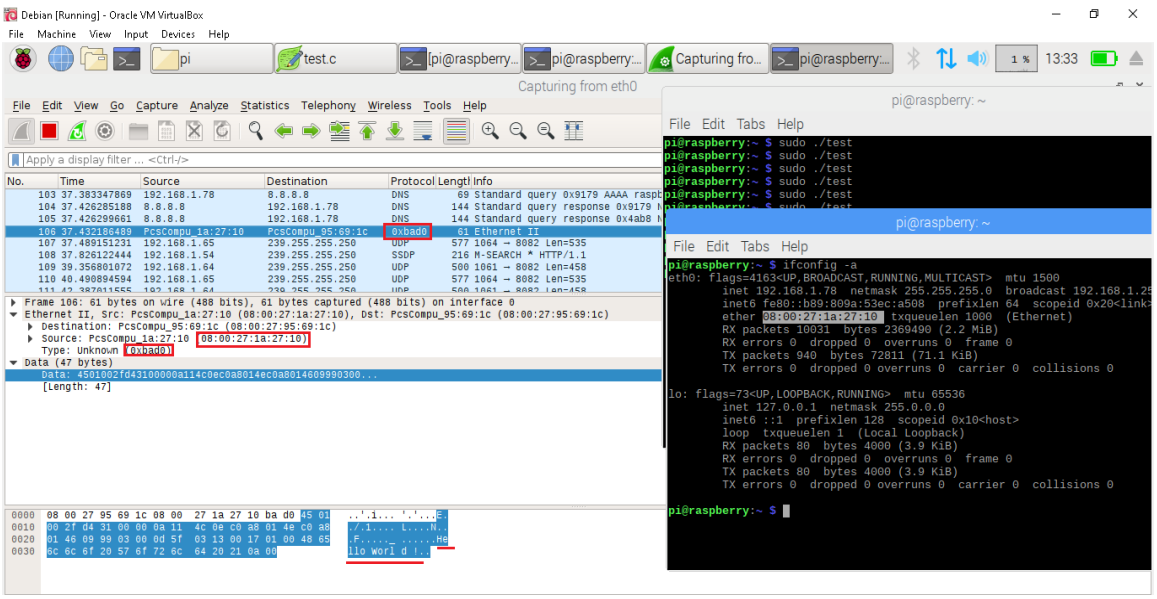


Figure 7: Screenshot of Sender system using Ethertype 0xBAD0

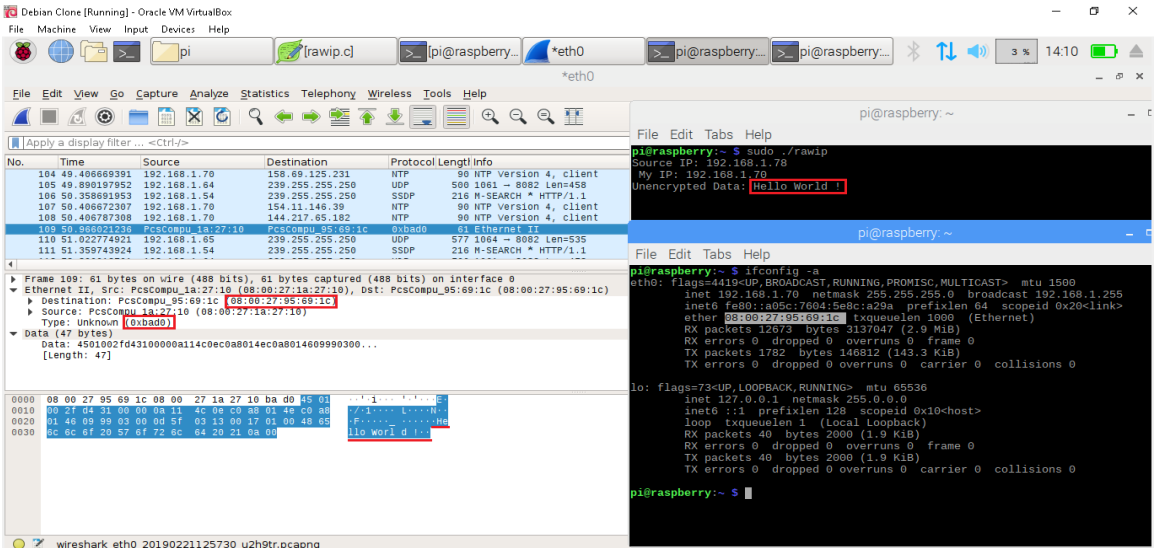


Figure 8: Screenshot of Receiver system using Ethertype 0xBAD0 and receiving data

STEP 3: CHANGE IP HEADER LENGTH AND VERSION

Once the Ethertype was successfully changed to 0xBAD0, IPv4 header length and version field were required to be changed. These changes in IP version and field length would help the packet from not getting accessed by attackers, as normal network adapter settings are programmed to understand packets with default IP packet version 4. To achieve this IPPROTO_RAW was used in the socket to define all custom IPv4 header fields including IP header length and IP version field.

STEP 4: CHANGE IP HEADER TOS FIELD

After defining the custom IP version and header length, the next step was to define the TOS field. Changing TOS services allowed us to use custom operations for the control systems like type of operation, criticality of operation, and data encryption. This required changes to be made in the Linux header files and in default TOS field mentioned in IPv4 header. To achieve this, contents of ip.h file were copied and using leafpad a new file ip3. h was created. The contents of ip.h were pasted in ip3.h file. All the further modifications for TOS services was made in ip3.h file. Changes in IP header file were made using below commands in command prompt of the sender:

```
sudo leafpad /usr/include/netinet/ip.h
```

copied contents of above file and pasted in

```
sudo leafpad /usr/include/netinet/ip3.h
```

Once the ip3.h file was created, it required file to be compiled, as changes were made in the header file. Updated ip3.h file was compiled using below commands ran in command prompt:

```
cd /usr/include/netinet
sudo gcc ip3.h
```

After creation of the ip3.h file, the default IP header file was replaced with the newly created ip3.h header file, in the sender and receiver programs.

From: `#include < netinet/ip.h >`

To: `#include < netinet/ip3.h >`

Once the header files were replaced in the sender and receiver programs, both the programs were saved and compiled. Also, it was verified that both sender and receiver systems were able to send and receive the data using updated IP header file.

STEP 5: CHANGE IP ADDRESS ASSIGNMENT

IP address field in the updated header structure was kept the same as default IP address field to make it compatible. However, it was parsed differently so that only sender and receiver of the packet can accept and process it. In the updated IP address field, first octet defines the geographic region that the device is operating in, second octet refers to the corporate subnet that the device is operating in and the third and fourth octets refers to the individual device itself. This was achieved by making changes in the Linux sub-Header file:

`#include <netinet/in.h>` which is present under main header file

`#include <arpa/inet.h>`

In the updated IPv4 packet, the first octet required all geographic regions to be defined. I have defined 247 Countries using first octet of IP address field. Also, octet being 8 bits long, it allowed upto 256 countries to be defined. The second octet required all Corporate subnets / Companies to be defined. I have defined 37 Companies in the modified second octet of IPv4 packet. These changes in the first and second octet are highlighted in the sender program. In the updated IPv4 packet, the third and fourth octet refers to the individual device itself. The third and fourth octets are defined using below code:

```
#define IN_CLASSC_HOST ((in_addr_t) & 0x0000ff00) = oct_three
#define IN_CLASSD_HOST ((in_addr_t) & 0x000000ff) = oct_four
```

Once this IPv4 octet changes were made in the header files, they were compiled as per below screenshot.

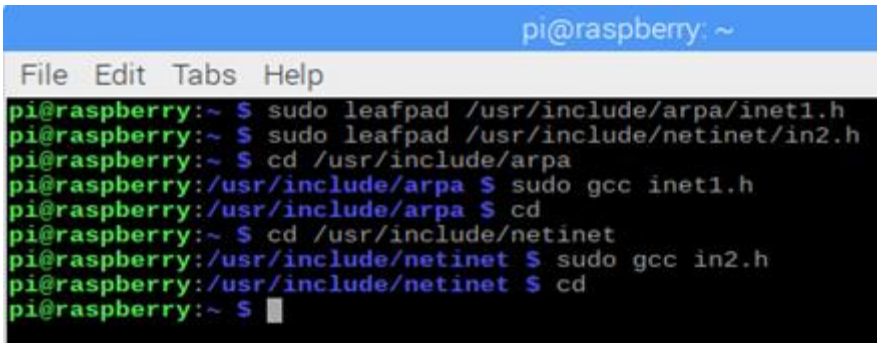


Figure 9: Compiling Header files

The same process was followed, and changes were made in the header files at the receiver end. After making changes, header files were compiled and successfully tested by sending and receiving data.

STEP 6: IMPLEMENTATION OF ENCRYPTION

Encryption field defined inside Option field, is controlled using TOS field. This field provides advantage of providing encryption through inbuild option field of the IPv4 packet rather than using separate encryption mechanisms like IPsec If the TOS field value is 0x14, it means operation requires encryption else for any other value of TOS, operation will be unencrypted. The shared secret keys used for encryption were generated using Diffie Hellman Algorithm. The encryption algorithm implemented is Caeser – Cipher and the constant value 0xE8E5, which is the hex value of secret key is added to the data for encryption. In the sender program, below values are used as Public and Private keys:

```
unsigned long long int Public_key1=63377;
unsigned long long int Public_key2=47339;
unsigned long long int Private_key=133;
```

The Public key generated using the Diffie Hellman code was required to be sent to the receiver end inside option field so that using Public Key, receiver would be able to generate the Diffie Hellman Secret key. This required option field value to be changed and the Linux OS had to be updated to make it understand that the value that is being carried in IP Header Option field is Diffie Hellman key value and not default option field value. This required changes to be made in the IP header file. Using section 4.22 of below RFC describing Experimental values for option field, we used the IPoption Type value as **158.**

https://tools.ietf.org/html/rfc7126

Further, the public key that was required to be sent to the receiver end is **39909**. So, the value of the IP option field in IP header file was set to:

```
#define IPOPT_SECURITY 0x39909059E
#define IPOPT_SEC IPOPT_SECURITY
```

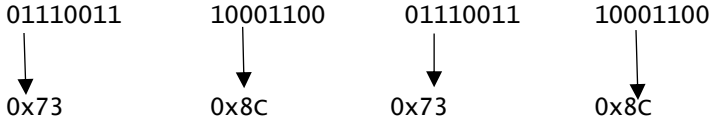
Breakdown of the Security value **0x39909059E** is:

9E = Hex value of Option Type 158.

05 = Total length of the option field including length field.

39909 = Public key that needs to be sent to the receiver.

Also, when the TOS field value was set to any other value instead of 0x14, it meant that operation was unencrypted and required below key value to be sent to the receiver:



Hence, for unencrypted operation value 0x738C738C was required to be sent to the receiver end using IP Option field. This was defined in IP option field using option value 158 as IPOPT_SECURITY1

```
#define IPOPT_SECURITY1 0x738C738C069E
#define IPOPT_SEC IPOPT_SECURITY1
```

After making these changes in IP Option field, the header file ip3.h was compiled at the sender end. Below screenshot shows compilation of ip3.h file

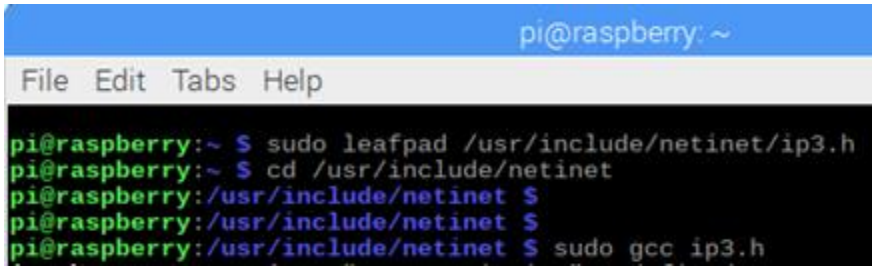


Figure 10: Compiling ip3.h file

Once the security values were defined, it was necessary to tell sender to select IPOPT_SECURITY when TOS field is 0x14 and IPOPT_SECURITY1 for any other value of TOS, this was achieved using below subpart of the program:

```
if(iph->tos == 0x14)
{
iph->opt = IPOPT_SECURITY;
}
else
{
iph->opt = IPOPT_SECURITY1;
}
```

To verify if these changes in header files were working, data "**Hello World!**" was sent over UDP protocol using port 787. So, if the TOS field is set to any value apart from 0x14, Receiver should print:

```
Unencrypted Data: Hello World!
Key: 738C738C (which was sent in IPOPT_SECURITY1)
```

Else,

if TOS value is set to 0x14, Receiver should print:

```
Encrypted Data: *********
Public Key: (Which was sent in IPOPT_SECURITY)
```

STEP 7: WORKING SCREENSHOTS

UNENCRYPTED OPERATION

Sender Configuration

Command Prompt showing program is running and TOS field value in IP Header is set to 0x11 in sender program.

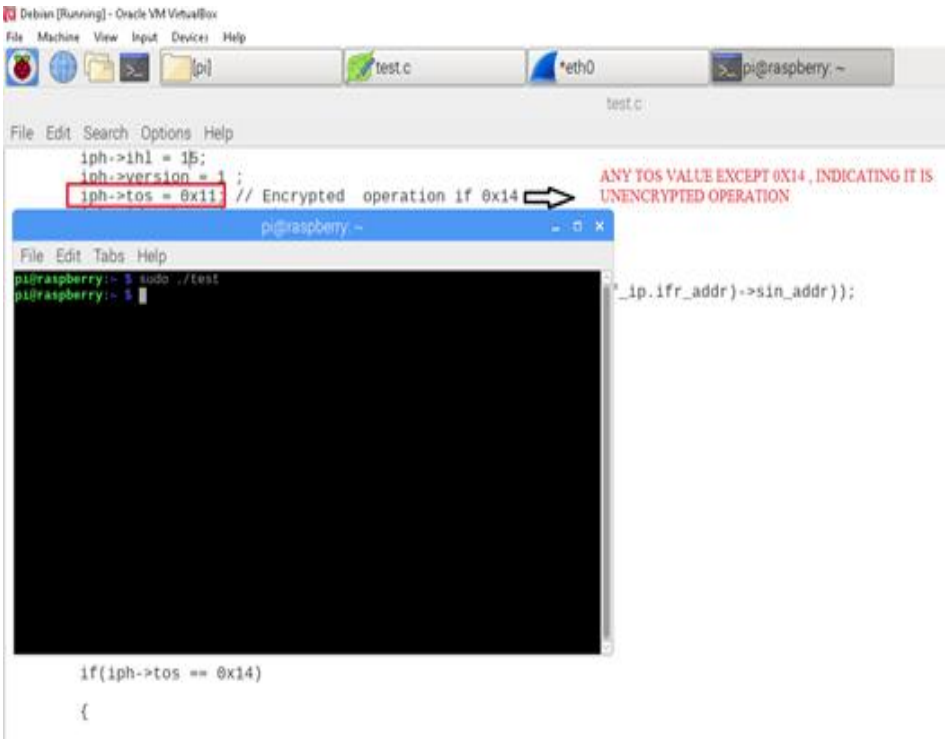


Figure 11: TOS value = 0x11 at Sender end for unencrypted operation

Wireshark showing custom Ethertype 0xBAD0 used to send unencrypted Data: Hello World!

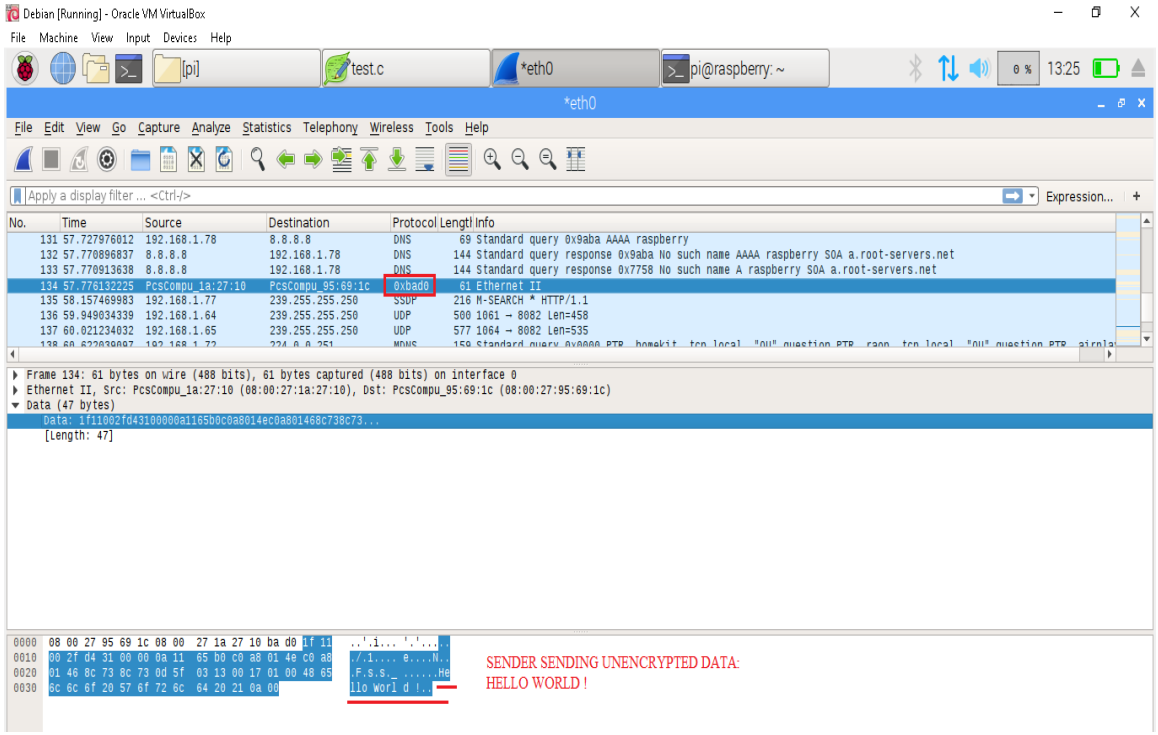


Figure 12: Wireshark showing data sent using custom IP Header

Receiver Configuration

Wireshark showing packets received with custom Ethertype 0xBAD0, with unencrypted data:

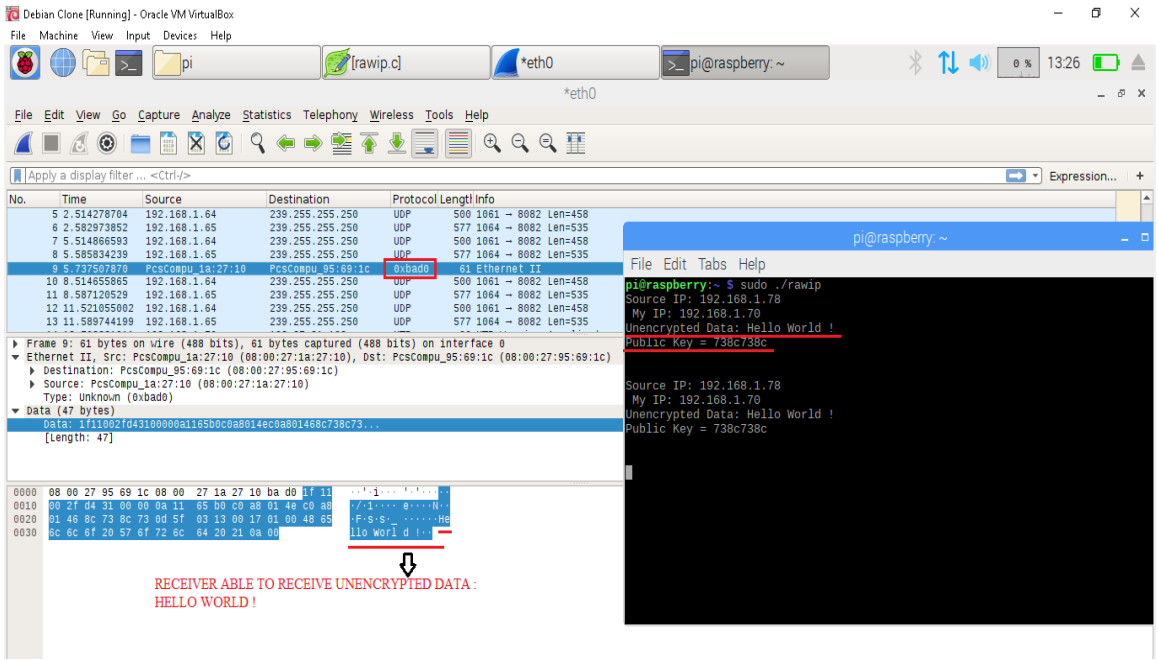


Figure 13: Wireshark showing data received using custom IP Header

ENCRYPTED OPERATION

Sender Configuration

Command Prompt showing program is running and sender program file showing TOS field value in IP header is 0x14.

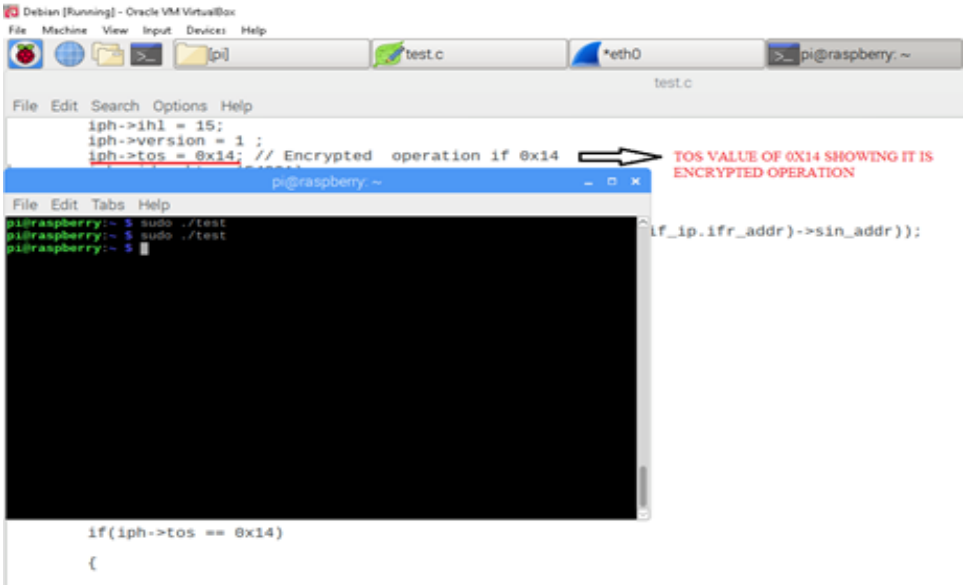


Figure 14: TOS value = 0x14 at Sender end for encrypted operation

Wireshark showing custom Ethertype 0xBAD0 used to send data: Hello World! which is Encrypted using the shared secret key of Diffie Hellman.

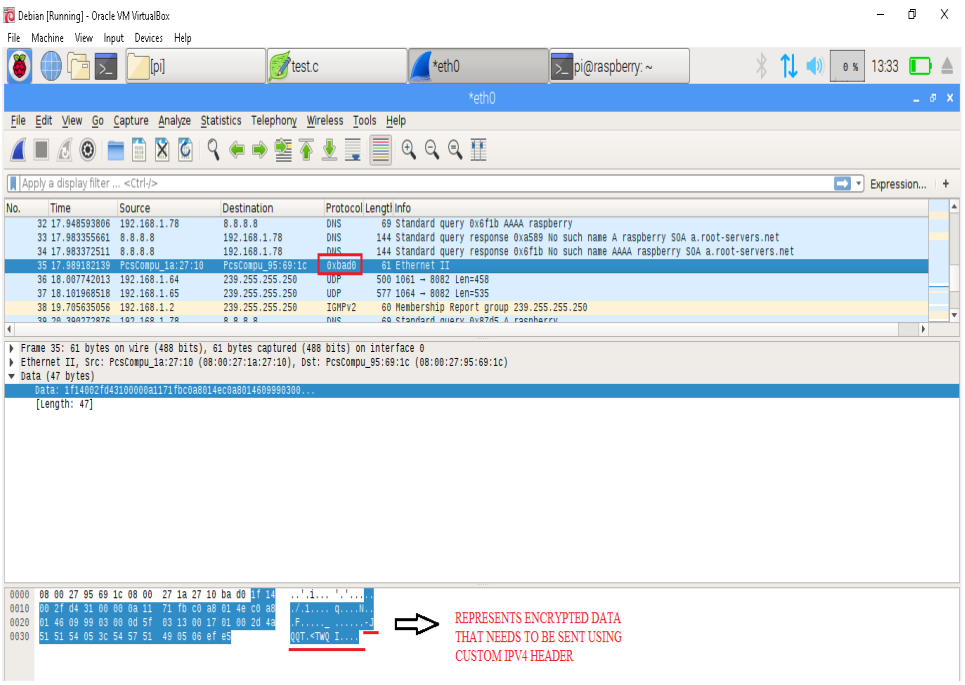


Figure 15: Wireshark showing Encrypted data sent

Receiver Configuration

Wireshark showing packets received with custom Ethertype 0xBAD0, with encrypted data that was sent by the sender and the value of Diffie Hellman Public Key.

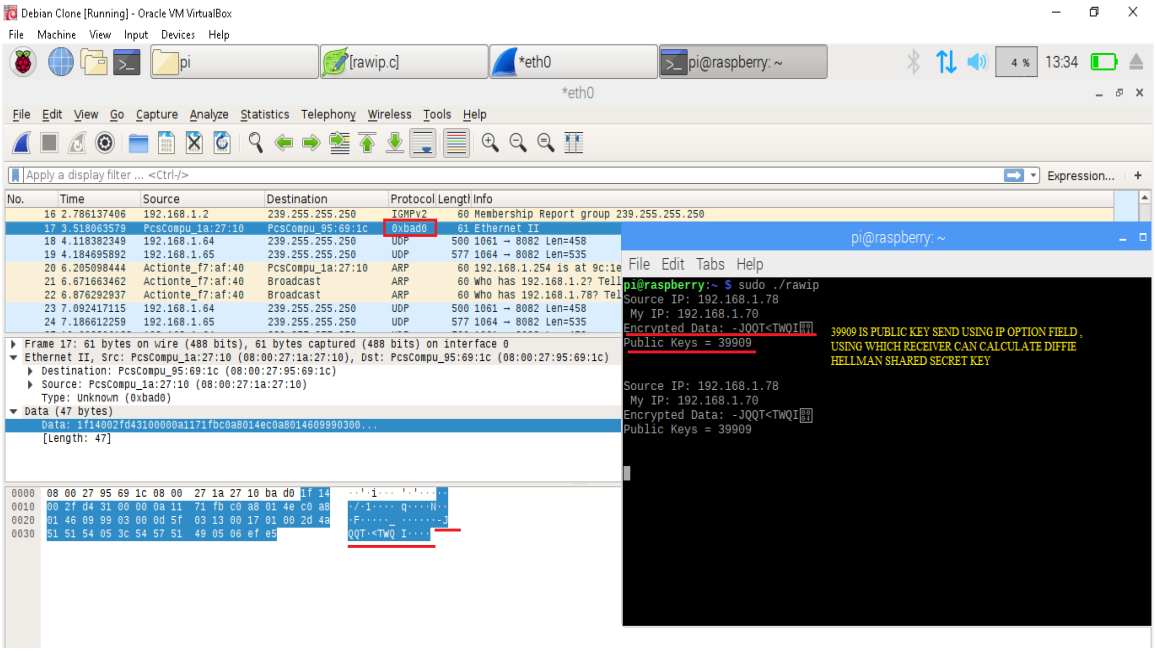


Figure 16: Wireshark showing Encrypted data received

Project Challenges

In the beginning of the project, it was required that Ethertype should have value 0xBAD0. To achieve this, I was making changes in the header file of the Raspberry Linux. However, it did not reflect as I was unaware that custom socket had to be defined which can carry custom IPv4 packets and Ethernet Frames. This was achieved using SOCK_RAW and IPPROTO_RAW using AF_SOCKET.

Initially, it was difficult to verify if the program was running correctly as Wireshark tool was only able to respond to the standard IPv4 packets and Ethernet frames of Ethertype 0x0800. Hence, to verify the working of program, data “Hello World!” was sent using custom IPv4 header and ethernet frames. By receiving the same data using custom Ethertype 0xBAD0, it was verified that required changes made to IPv4 Header and Ethernet type were working.

Defining custom IP option field was challenging as the default RFC 791 for IPv4 does not mention any IP option type that can be used for experimental purpose. However, Professor Rogers helped me by guiding RFC 7126 and pointing me to section 4.22 of Page 27 and explained me that we can use Experimental Option Type 158.

Renegotiation of Encryption keys was not possible as the renegotiated/updated key was required to be sent using IP Option Field. However, the IP Option field was given static value using Experimental option field 158 in ip3 header file. So, every time for renegotiation, the IP header file was required to be defined with new values of IP Option field and required to be compiled as per the renegotiation time which was not feasible.

Conclusion

This project can be used for improving security of data as it changes default IP header format and provides inbuilt encryption in IP header itself along with TOS field providing several operation and control functions like requirement of encryption, if the operation is manual or automatic, criticality of operation.

There are few improvements that can be made to make this project more scalable. Custom sockets only allow to define a specific Transport Layer Protocol, either TCP or UDP but not both. Hence, in this project we have used UDP protocol for both communication and transportation of data. Security can be further improved if some way is found out to establish connection between two systems using TCP and once connection is made, data is sent over UDP.

Also, as mentioned in the last point of Project challenge section, renegotiation of encryption key was not performed in this project. If some solution is found to provide renegotiation of secret key after specific time interval, security of data can be further enhanced.

Appendix A

Sender main Program

```
#include <arpa/inet1.h>
#include <linux/if_packet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/ether.h>
#include <netinet/ip3.h>
#include <netinet/udp.h>
#include <net/ethernet.h>
#define MY_DEST_MAC0 0x08
#define MY_DEST_MAC1 0x00
#define MY_DEST_MAC2 0x27
#define MY_DEST_MAC3 0x95
#define MY_DEST_MAC4 0x69
#define MY_DEST_MAC5 0x1C
#define DEFAULT_IF "eth0"
#define BUF_SIZ 1024
/* Diffie Hellman */
unsigned long long int power (unsigned int a,unsigned int b,unsigned int mod)
{
unsigned long long int t;
if(b==1)
return a;
t=power(a,b/2,mod);
if(b%2==0)
return (t*t)%mod;
else
return (((t*t)%mod)*a)%mod;
}
```

```
int main (int argc, char *argv[])
{
int sockfd;
struct ifreq if_idx;
struct ifreq if_mac;
struct ifreq if_ip;
int tx_len = 0;
int ttl = 10;
char sendbuf[1024];
unsigned long long int Public_key1=63377;
unsigned long long int Public_key2=47339;
unsigned long long int Private_key=133;
unsigned long long int x,secret;
x =power(Public_key2, Private_key, Public_key1);
secret =power(x,Private_key,Public_key1);
struct ether_header *eh = (struct ether_header *) sendbuf;
struct iphdr *iph = (struct iphdr *) (sendbuf + sizeof(struct ether_header));
struct udphdr *udph = (struct udphdr *) (sendbuf + sizeof(struct iphdr) +
sizeof(struct ether_header));
struct sockaddr_ll socket_address;
char ifName[IFNAMSIZ];
/* Get interface name */
if (argc > 1)
strcpy(ifName, argv[1]);
else
strcpy(ifName, DEFAULT_IF);
/* Open RAW socket to send on */
if ((sockfd = socket(AF_PACKET, SOCK_RAW, IPPROTO_RAW)) == -1) {
perror("socket");
}
/* Get the index of the interface to send on */
memset(&if_idx, 0, sizeof(struct ifreq));
strncpy(if_idx.ifr_name, "eth0", IFNAMSIZ-1);
if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0)
perror("SIOCGIFINDEX");
/* Get the MAC address of the interface to send on */
memset(&if_mac, 0, sizeof(struct ifreq));
```

```
strncpy(if_mac.ifr_name, "eth0", IFNAMSIZ-1);
if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0)
perror("SIOCGIFHWADDR");
/* Get the IP address of the interface to send on */
memset(&if_ip, 0, sizeof(struct ifreq));
strncpy(if_ip.ifr_name, "eth0", IFNAMSIZ-1);
if (ioctl(sockfd, SIOCGIFADDR, &if_ip) < 0)
perror("SIOCGIFADDR");
/* Construct the Ethernet header */
memset(sendbuf, 0, 1024);
/* Ethernet header */
eh->ether_shost[0] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[0];
eh->ether_shost[1] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[1];
eh->ether_shost[2] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[2];
eh->ether_shost[3] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[3];
eh->ether_shost[4] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[4];
eh->ether_shost[5] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[5];
eh->ether_dhost[0] = MY_DEST_MAC0;
eh->ether_dhost[1] = MY_DEST_MAC1;
eh->ether_dhost[2] = MY_DEST_MAC2;
eh->ether_dhost[3] = MY_DEST_MAC3;
eh->ether_dhost[4] = MY_DEST_MAC4;
eh->ether_dhost[5] = MY_DEST_MAC5;
/* Ethertype field */
eh->ether_type = htons(0xBAD0);
tx_len += sizeof(struct ether_header);
/* IP Header */
iph->ihl = 15;
iph->version = 1;
iph->tos = 0x14; // Encrypted operation if 0x14
iph->id = htons(54321);
iph->ttl = ttl;//hops
iph->protocol = 17; // UDP
iph->saddr    =    inet_addr(inet_ntoa(((struct    sockaddr_in    *)&if_ip.ifr_addr)-
>sin_addr));
//iph->saddr = inet_addr("192.168.1.78");
/* Destination IP address */
```

```
iph->daddr = inet_addr("192.168.1.70");
tx_len += sizeof(struct iphdr);
/* UDP Header */
udph->source = htons(3423);
udph->dest = htons(787);
udph->check = 1; // skip
tx_len += sizeof(struct udphdr);
if(iph->tos == 0x14)
{
iph->opt = IPOPT_SECURITY >> 16;
sendbuf[tx_len++] = 0x48 + 0xE8E5;
sendbuf[tx_len++] = 0x65 + 0xE8E5;
sendbuf[tx_len++] = 0x6c + 0xE8E5;
sendbuf[tx_len++] = 0x6c + 0xE8E5;
sendbuf[tx_len++] = 0x6f + 0xE8E5;
sendbuf[tx_len++] = 0x20 + 0xE8E5;
sendbuf[tx_len++] = 0x57 + 0xE8E5;
sendbuf[tx_len++] = 0x6f + 0xE8E5;
sendbuf[tx_len++] = 0x72 + 0xE8E5;
sendbuf[tx_len++] = 0x6c + 0xE8E5;
sendbuf[tx_len++] = 0x64 + 0xE8E5;
sendbuf[tx_len++] = 0x20 + 0xE8E5;
sendbuf[tx_len++] = 0x21 + 0xE8E5;
sendbuf[tx_len++] = 0x0a + 0xE8E5;
sendbuf[tx_len++] = 0x00 + 0xE8E5;
}
else
{
iph->opt = IPOPT_SECURITY1 >> 16;
sendbuf[tx_len++] = 0x48;
sendbuf[tx_len++] = 0x65;
sendbuf[tx_len++] = 0x6c;
sendbuf[tx_len++] = 0x6c;
sendbuf[tx_len++] = 0x6f;
sendbuf[tx_len++] = 0x20;
sendbuf[tx_len++] = 0x57;
```

```
sendbuf[tx_len++] = 0x6f;
sendbuf[tx_len++] = 0x72;
sendbuf[tx_len++] = 0x6c;
sendbuf[tx_len++] = 0x64;
sendbuf[tx_len++] = 0x20;
sendbuf[tx_len++] = 0x21;
sendbuf[tx_len++] = 0x0a;
sendbuf[tx_len++] = 0x00;
}
unsigned short csum(unsigned short *buf, int nwords)
{
unsigned long sum;
for(sum=0; nwords>0; nwords--)
sum += (*buf++);
sum = (sum >> 16) + (sum &0xffff);
sum += (sum >> 16);
return (unsigned short)(~sum);
}
/* Length of UDP payload and header */
udph->len = htons(tx_len - sizeof(struct ether_header) - sizeof(struct iphdr));
/* Length of IP payload and header
/* Length of IP payload and header */
iph->tot_len = htons(tx_len - sizeof(struct ether_header));
/* Calculate IP checksum on completed header */
iph->check   =   csum((unsigned   short   *)(sendbuf+sizeof(struct   ether_header)),
sizeof(struct iphdr)/2);
/* Index of the network device */
socket_address.sll_ifindex = if_idx.ifr_ifindex;
/* Address length*/
socket_address.sll_halen = ETH_ALEN;
/* Destination MAC */
socket_address.sll_addr[0] = MY_DEST_MAC0;
socket_address.sll_addr[1] = MY_DEST_MAC1;
socket_address.sll_addr[2] = MY_DEST_MAC2;
socket_address.sll_addr[3] = MY_DEST_MAC3;
socket_address.sll_addr[4] = MY_DEST_MAC4;
socket_address.sll_addr[5] = MY_DEST_MAC5;
```

```
if (sendto(sockfd, sendbuf, tx_len, 0, (struct sockaddr*)&socket_address,
sizeof(struct sockaddr_ll)) < 0)
printf("Send failed\n");
return 0;
}
```

Sender ip3.h Program (Changes made to the default ip.h file are highlighted).

```
#ifndef __NETINET_IP_H
#define __NETINET_IP_H 1
#include <features.h>
#include <sys/types.h>
#include <netinet/in.h>
__BEGIN_DECLS
struct timestamp
{
u_int8_t len;
u_int8_t ptr;
#if __BYTE_ORDER == __LITTLE_ENDIAN
unsigned int flags:4;
unsigned int overflow:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
unsigned int overflow:4;
unsigned int flags:4;
#else
# error "Please fix <bits/endian.h>"
#endif
u_int32_t data[9];
};
struct iphdr
{
#if __BYTE_ORDER == __LITTLE_ENDIAN
unsigned int ihl:4;
unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
unsigned int version:4;
unsigned int ihl:4;
```

```
#else
# error "Please fix <bits/endian.h>"
#endif
u_int8_t tos;
u_int16_t tot_len;
u_int16_t id;
u_int16_t frag_off;
u_int8_t ttl;
u_int8_t protocol;
u_int16_t check;
u_int32_t saddr;
u_int32_t daddr;
u_int32_t opt;
};
struct ip
{
#if __BYTE_ORDER == __LITTLE_ENDIAN
unsigned int ip_hl:4; /* header length */
unsigned int ip_v:4; /* version */
#endif
#if __BYTE_ORDER == __BIG_ENDIAN
unsigned int ip_v:4; /* version */
unsigned int ip_hl:4; /* header length */
#endif
u_int8_t ip_tos; /* type of service */
u_short ip_len; /* total length */
u_short ip_id; /* identification */
u_short ip_off; /* fragment offset field */
#define IP_RF 0x8000 /* reserved fragment flag */
#define IP_DF 0x4000 /* dont fragment flag */
#define IP_MF 0x2000
#define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
u_int8_t ip_ttl; /* time to live */
u_int8_t ip_p; /* protocol */
u_short ip_sum; /* checksum */
u_int32_t ip_opt;
```

```
/*u_short ip_optlen;
u_short ip_opt_data;*/
struct in_addr ip_src, ip_dst; /* source and dest address */
#define IPOPT_SECURITY 0x39909059E /* provide s,c,h,tcc     */
#define IPOPT_SECURITY1         0x738C738C069E
#define IPOPT_SEC IPOPT_SECURITY
#define IPOPT_SEC IPOPT_SECURITY1
};
struct ip_timestamp
{
u_int8_t ipt_code; /* IPOPT_TS */
u_int8_t ipt_len; /* size of structure (variable) */
u_int8_t ipt_ptr; /* index of current entry */
#if __BYTE_ORDER == __LITTLE_ENDIAN
unsigned int ipt_flg:4; /* flags, see below */
unsigned int ipt_oflw:4; /* overflow counter */
#endif
#if __BYTE_ORDER == __BIG_ENDIAN
unsigned int ipt_oflw:4; /* overflow counter */
unsigned int ipt_flg:4; /* flags, see below */
#endif
u_int32_t data[9];
};
#endif /* __USE_MISC */
#define IPVERSION 1               /* IP version number */
#define IP_MAXPACKET 65535 /* maximum packet size */
#define IPTOS_Normal Operation 0x00
#define IPTOS_Immediate Operational Shutdown 0x01
#define IPTOS_Routine Operation Response 0x02
#define IPTOS_Priority Operation Response 0x03
#define IPTOS_Critical Operation Response 0x04
#define IPTOS_Manual Control/Emergency Take Over0x05
#define IPTOS_Regular Data Reporting 0x06
#define IPTOS_Priority Data Reporting 0x07
#define IPTOS_Immediate Data Reporting 0x08
#define IPTOS_Streaming Data Reporting 0x09
```

```
#define IPTOS_Perform System Check 0x10
#define IPTOS_Report System Status 0x11
#define IPTOS_Immediate Report In-Return Home 0x12
#define IPTOS_Encrypted Operation/ Renegotiate 0x13
#define IPTOS_Encrypted Operation/ Normal 0x14
#define IPOPT_COPY 0x80
#define IPOPT_CLASS_MASK 0x60
#define IPOPT_NUMBER_MASK 0x1f
#define IPOPT_COPIED(o) ((o) & IPOPT_COPY)
#define IPOPT_CLASS(o) ((o) & IPOPT_CLASS_MASK)
#define IPOPT_NUMBER(o) ((o) & IPOPT_NUMBER_MASK)
#define IPOPT_CONTROL 0x00
#define IPOPT_RESERVED1 0x20
#define IPOPT_DEBMEAS 0x40
#define IPOPT_MEASUREMENT IPOPT_DEBMEAS
#define IPOPT_RESERVED2 0x60
#define MAXTTL 255 /* maximum time to live (seconds) */
#define IPDEFTTL 64 /* default ttl, from RFC 1340 */
#define IPFRAGTTL 60 /* time to live for frags, slowhz */
#define IPTTLDEC 1 /* subtracted when forwarding */
#define IP_MSS 576 /* default maximum segment size */
__END_DECLS
#endif /* netinet/ip.h */
```

Sender in2.h Program (This header file is present in the #include <arpa/inet1.h>. Changes made to the default ip.h file are highlighted).

```
#ifndef _NETINET_IN_H
#define _NETINET_IN_H 1
#include <features.h>
#include <stdint.h>
#include <sys/socket1.h>
#include <bits/types.h>
#include <stdlib.h>
__BEGIN_DECLS
/* Internet address.  */
typedef uint32_t in_addr_t;
```

```
uint32_t oct_one;
uint32_t oct_two;
uint32_t oct_three;
uint32_t oct_four;
typedef uint16_t in_port_t;
struct in_addr
{
in_addr_t s_addr;
};
/* Get system-specific definitions.*/
#include <bits/in.h>
/* Standard well-defined IP protocols.*/
enum
{
IPPROTO_IP = 0,/* Dummy protocol for TCP.*/
#define IPPROTO_IP IPPROTO_IP
IPPROTO_ICMP = 1,/* Internet Control Message Protocol*/
#define IPPROTO_ICMP IPPROTO_ICMP
IPPROTO_IGMP = 2,/* Internet Group Management Protocol. */
#define IPPROTO_IGMP IPPROTO_IGMP
IPPROTO_IPIP = 4,/* IPIP tunnels (older KA9Q tunnels use 94).*/
#define IPPROTO_IPIP IPPROTO_IPIP
IPPROTO_TCP = 6,/* Transmission Control Protocol.  */
#define IPPROTO_TCP IPPROTO_TCP
IPPROTO_EGP = 8,/* Exterior Gateway Protocol.*/
#define IPPROTO_EGP IPPROTO_EGP
IPPROTO_PUP = 12,/* PUP protocol.*/
#define IPPROTO_PUP IPPROTO_PUP
IPPROTO_UDP = 17,/* User Datagram Protocol.*/
#define IPPROTO_UDP IPPROTO_UDP
IPPROTO_IDP = 22,/* XNS IDP protocol.*/
#define IPPROTO_IDP IPPROTO_IDP
IPPROTO_TP = 29,/* SO Transport Protocol Class 4*/
#define IPPROTO_TP IPPROTO_TP
IPPROTO_DCCP = 33,/* Datagram Congestion Control Protocol*/
#define IPPROTO_DCCP IPPROTO_DCCP
```

```
IPPROTO_IPV6 = 41,/* IPv6 header.*/
#define IPPROTO_IPV6 IPPROTO_IPV6
IPPROTO_RSVP = 46,/* Reservation Protocol.*/
#define IPPROTO_RSVP IPPROTO_RSVP
IPPROTO_GRE = 47,/* General Routing Encapsulation.*/
#define IPPROTO_GRE IPPROTO_GRE
IPPROTO_ESP = 50,/* encapsulating security payload.*/
#define IPPROTO_ESP IPPROTO_ESP
IPPROTO_AH = 51,/* authentication header.*/
#define IPPROTO_AH IPPROTO_AH
IPPROTO_MTP = 92,/* Multicast Transport Protocol.*/
#define IPPROTO_MTP IPPROTO_MTP
IPPROTO_BEETPH = 94,/* IP option pseudo header for BEET.*/
#define IPPROTO_BEETPH IPPROTO_BEETPH
IPPROTO_ENCAP = 98,/* Encapsulation Header.  */
#define IPPROTO_ENCAP IPPROTO_ENCAP
IPPROTO_PIM = 103,/* Protocol Independent Multicast.*/
#define IPPROTO_PIM IPPROTO_PIM
IPPROTO_COMP = 108,/* Compression Header Protocol.  */
#define IPPROTO_COMP IPPROTO_COMP
IPPROTO_SCTP = 132,/* Stream Control Transmission Protocol.*/
#define IPPROTO_SCTP IPPROTO_SCTP
IPPROTO_UDPLITE = 136, /* UDP-Lite protocol.*/
#define IPPROTO_UDPLITE IPPROTO_UDPLITE
IPPROTO_MPLS = 137,/* MPLS in IP.  */
#define IPPROTO_MPLS IPPROTO_MPLS
IPPROTO_RAW = 255,/* Raw IP packets.*/
#define IPPROTO_RAW IPPROTO_RAW
IPPROTO_MAX
};
/* If __USE_KERNEL_IPV6_DEFS is 1 then the user has included the kernel
   network headers first and we should use those ABI-identical definitions
   instead of our own, otherwise 0.  */
#if !__USE_KERNEL_IPV6_DEFS
enum
{
```

```
IPPROTO_HOPOPTS = 0,/* IPv6 Hop-by-Hop options.  */
#define IPPROTO_HOPOPTS IPPROTO_HOPOPTS
IPPROTO_ROUTING = 43,/* IPv6 routing header.  */
#define IPPROTO_ROUTING IPPROTO_ROUTING
IPPROTO_FRAGMENT = 44, /* IPv6 fragmentation header.  */
#define IPPROTO_FRAGMENT IPPROTO_FRAGMENT
IPPROTO_ICMPV6 = 58,/* ICMPv6.  */
#define IPPROTO_ICMPV6 IPPROTO_ICMPV6
IPPROTO_NONE = 59, /* IPv6 no next header.  */
#define IPPROTO_NONE IPPROTO_NONE
IPPROTO_DSTOPTS = 60,/* IPv6 destination options.  */
#define IPPROTO_DSTOPTS IPPROTO_DSTOPTS
IPPROTO_MH = 135 /* IPv6 mobility header.  */
#define IPPROTO_MH IPPROTO_MH
};
#endif /* !__USE_KERNEL_IPV6_DEFS */
enum
IPPORT_ECHO = 7, /* Echo service.  */
IPPORT_DISCARD = 9, /* Discard transmissions service.  */
IPPORT_SYSTAT = 11, /* System status service.  */
IPPORT_DAYTIME = 13, /* Time of day service.  */
IPPORT_NETSTAT = 15, /* Network status service.  */
IPPORT_FTP = 21, /* File Transfer Protocol.  */
IPPORT_TELNET = 23, /* Telnet protocol.  */
IPPORT_SMTP = 25, /* Simple Mail Transfer Protocol.  */
IPPORT_TIMESERVER = 37, /* Timeserver service.  */
IPPORT_NAMESERVER = 42, /* Domain Name Service.  */
IPPORT_WHOIS = 43, /* Internet Whois service.  */
IPPORT_MTP = 57,
IPPORT_TFTP = 69, /* Trivial File Transfer Protocol.  */
IPPORT_RJE = 77,
IPPORT_FINGER = 79, /* Finger service.  */
IPPORT_TTYLINK = 87,
IPPORT_SUPDUP = 95, /* SUPDUP protocol.  */
IPPORT_EXECSERVER = 512, /* execd service.  */
IPPORT_LOGINSERVER = 513, /* rlogind service.  */
```

```
IPPORT_CMDSERVER = 514,
IPPORT_EFSSERVER = 520,
/* UDP ports.  */
IPPORT_BIFFUDP = 512,
IPPORT_WHOSERVER = 513,
IPPORT_ROUTESERVER = 520,
/* Ports less than this value are reserved for privileged processes.  */
IPPORT_RESERVED = 1024,
/* Ports greater this value are reserved for (non-privileged) servers.  */
IPPORT_USERRESERVED = 5000
};
void test ()
{
#define IN_CLASSA_HOST ((in_addr_t) & 0xff000000) = oct_one
switch(oct_one)
{
case 1:
oct_one = atoi("Afghanistan");
break;
case 2:
oct_one = atoi("Åland Islands");
break;
case 3:
oct_one = atoi("Albania");
break;
case 4:
oct_one = atoi("Algeria");
break;
case 5:
oct_one = atoi("American Samoa");
break;
case 6:
oct_one = atoi("Andorra");
break;
case 7:
oct_one = atoi("Angola");
```

```
break;
case 8:
oct_one = atoi("Anguilla");
break;
case 9:
oct_one = atoi("Antigua and Barbuda");
break;
case 10:
oct_one = atoi("Argentina");
break;
case 11:
oct_one = atoi("Armenia");
break;
case 12:
oct_one = atoi("Aruba");
break;
case 13:
oct_one = atoi("Australia");
break;
case 14:
oct_one = atoi("Austria");
break;
case 15:
oct_one = atoi("Azerbaijan");
break;
case 16:
oct_one = atoi("Bahamas");
break;
case 17:
oct_one = atoi("Bahrain");
break;
case 18:
oct_one = atoi("Bangladesh");
break;
case 19:
oct_one = atoi("Barbados");
```

```
break;
case 20:
oct_one = atoi("Belarus");
break;
case 21:
oct_one = atoi("Belgium");
break;
case 22:
oct_one = atoi("Belize");
break;
case 23:
oct_one = atoi("Benin");
break;
case 24:
oct_one = atoi("Bermuda");
break;
case 25:
oct_one = atoi("Bhutan");
break;
case 26:
oct_one = atoi("Bolivia");
break;
case 27:
oct_one = atoi("Bosnia and Herzegovina");
break;
case 28:
oct_one = atoi("Botswana");
break;
case 29:
oct_one = atoi("Brazil");
break;
case 30:
oct_one = atoi("British Antarctic Territory");
break;
case 31:
oct_one = atoi("British Indian Ocean Territory");
```

```
break;
case 32:
oct_one = atoi("British Virgin Islands");
break;
case 33:
oct_one = atoi("Brunei");
break;
case 34:
oct_one = atoi("Bulgaria");
break;
case 35:
oct_one = atoi("Burkina Faso");
break;
case 36:
oct_one = atoi("Burundi");
break;
case 37:
oct_one = atoi("Cambodia");
break;
case 38:
oct_one = atoi("Cameroon");
break;
case 39:
oct_one = atoi("Canada");
break;
case 40:
oct_one = atoi("Cape Verde");
break;
case 41:
oct_one = atoi("Cayman Islands");
break;
case 42:
oct_one = atoi("Central African Republic");
break;
case 43:
oct_one = atoi("Chad");
```

```
break;
case 44:
oct_one = atoi("Chile");
break;
case 45:
oct_one = atoi("China");
break;
case 46:
oct_one = atoi("Christmas Island");
break;
case 47:
oct_one = atoi("Cocos Islands");
break;
case 48:
oct_one = atoi("Colombia");
break;
case 49:
oct_one = atoi("Comoros");
break;
case 50:
oct_one = atoi("Congo");
break;
case 51:
oct_one = atoi("Congo, Democratic Republic");
break;
case 52:
oct_one = atoi("Cook Islands");
break;
case 53:
oct_one = atoi("Curacao");
break;
case 54:
oct_one = atoi("Cyprus");
break;
case 55:
oct_one = atoi("Costa Rica");
```

```
break;
case 56:
oct_one = atoi("Côte d'Ivoire ");
break;
case 57:
oct_one = atoi("Croatia");
break;
case 58:
oct_one = atoi("Cuba");
break;
case 59:
oct_one = atoi("Curaçao");
break;
case 60:
oct_one = atoi("Cyprus");
break;
case 61:
oct_one = atoi("Czech Republic");
break;
case 62:
oct_one = atoi("Denmark");
break;
case 63:
oct_one = atoi("Djibouti");
break;
case 64:
oct_one = atoi("Dominica");
break;
case 65:
oct_one = atoi("Dominican Republic");
break;
case 66:
oct_one = atoi("East Timor");
break;
case 67:
oct_one = atoi("Ecuador");
```

```
break;
case 68:
oct_one = atoi("El Salvador");
break;
case 69:
oct_one = atoi("Egypt");
break;
case 70:
oct_one = atoi("Equatorial Guinea");
break;
case 71:
oct_one = atoi("Eritrea");
break;
case 72:
oct_one = atoi("Estonia");
break;
case 73:
oct_one = atoi("Ethiopia");
break;
case 74:
oct_one = atoi("Falkland Islands");
break;
case 75:
oct_one = atoi("Faroe Islands");
break;
case 76:
oct_one = atoi("Fiji");
break;
case 77:
oct_one = atoi("Finland");
break;
case 78:
oct_one = atoi("France");
break;
case 79:
oct_one = atoi("French Guiana");
```

```
break;
case 80:
oct_one = atoi("French Polynesia");
break;
case 81:
oct_one = atoi("Czech French Southern and Antarctic Territories");
break;
case 82:
oct_one = atoi("Gabon");
break;
case 83:
oct_one = atoi("Gambia");
break;
case 84:
oct_one = atoi("Georgia");
break;
case 85:
oct_one = atoi("Germany");
break;
case 86:
oct_one = atoi("Ghana");
break;
case 87:
oct_one = atoi("Gibraltar");
break;
case 88:
oct_one = atoi("Greece");
break;
case 89:
oct_one = atoi("Greenland");
break;
case 90:
oct_one = atoi("Grenada");
break;
case 91:
oct_one = atoi("Guadeloupe");
```

```
break;
case 92:
oct_one = atoi("Guam");
break;
case 93:
oct_one = atoi("Guatemala");
break;
case 94:
oct_one = atoi("Guernsey");
break;
case 95:
oct_one = atoi("Guinea");
break;
case 96:
oct_one = atoi("Guinea Bissau");
break;
case 97:
oct_one = atoi("Guyana");
break;
case 98:
oct_one = atoi("Haiti");
break;
case 99:
oct_one = atoi("Heard and McDonald Islands");
break;
case 100:
oct_one = atoi("Honduras");
break;
case 101:
oct_one = atoi("Hong Kong");
break;
case 102:
oct_one = atoi("Hungary");
break;
case 103:
oct_one = atoi("Iceland");
```

```
break;
case 104:
oct_one = atoi("India");
break;
case 105:
oct_one = atoi("Indonesia");
break;
case 106:
oct_one = atoi("Iran");
break;
case 107:
oct_one = atoi("Iraq");
break;
case 108:
oct_one = atoi("Ireland");
break;
case 109:
oct_one = atoi("Isle of Man");
break;
case 110:
oct_one = atoi("Israel");
break;
case 111:
oct_one = atoi("Italy");
break;
case 112:
oct_one = atoi("Jamaica");
break;
case 113:
oct_one = atoi("Japan");
break;
case 114:
oct_one = atoi("Jersey");
break;
case 115:
oct_one = atoi("Jordan");
```

```
break;
case 116:
oct_one = atoi("Kazakhstan");
break;
case 117:
oct_one = atoi("Kenya");
break;
case 118:
oct_one = atoi("Kiribati");
break;
case 119:
oct_one = atoi("Korea, North");
break;
case 120:
oct_one = atoi("Korea, South");
break;
case 121:
oct_one = atoi("Kosovo");
break;
case 122:
oct_one = atoi("Kuwait");
break;
case 123:
oct_one = atoi("Kyrgyzstan");
break;
case 124:
oct_one = atoi("Laos");
break;
case 125:
oct_one = atoi("Latvia");
break;
case 126:
oct_one = atoi("Lebanon");
break;
case 127:
oct_one = atoi("Lesotho");
```

```
break;
case 128:
oct_one = atoi("Liberia");
break;
case 129:
oct_one = atoi("Libya");
break;
case 130:
oct_one = atoi("Liechtenstein");
break;
case 131:
oct_one = atoi("Lithuania");
break;
case 132:
oct_one = atoi("Luxembourg");
break;
case 133:
oct_one = atoi("Macau");
break;
case 134:
oct_one = atoi("Macedonia");
break;
case 135:
oct_one = atoi("Madagascar");
break;
case 136:
oct_one = atoi("Malawi");
break;
case 137:
oct_one = atoi("Malaysia");
break;
case 138:
oct_one = atoi("Maldives");
break;
case 139:
oct_one = atoi("Mali");
```

```
break;
case 140:
oct_one = atoi("Malta");
break;
case 141:
oct_one = atoi("Marshall Islands");
break;
case 142:
oct_one = atoi("Mauritania");
break;
case 143:
oct_one = atoi("Mauritius");
break;
case 144:
oct_one = atoi("Martinique");
break;
case 145:
oct_one = atoi("Mayotte");
break;
case 146:
oct_one = atoi("Mexico");
break;
case 147:
oct_one = atoi("Micronesia");
break;
case 148:
oct_one = atoi("Moldova");
break;
case 149:
oct_one = atoi("Monaco");
break
case 150:
oct_one = atoi("Mongolia");
break;
case 151:
oct_one = atoi("Montenegro");
```

```
break;
case 152:
oct_one = atoi("Montserrat");
break;
case 153:
oct_one = atoi("Morocco");
break;
case 154:
oct_one = atoi("Mozambique");
break;
case 155:
oct_one = atoi("Myanmar");
break;
case 156:
oct_one = atoi("Namibia");
break;
case 157:
oct_one = atoi("Nauru");
break;
case 158:
oct_one = atoi("Nepal");
break;
case 159:
oct_one = atoi("Netherlands");
break;
case 160:
oct_one = atoi("New Caledonia");
break;
case 161:
oct_one = atoi("New Zealand");
break;
case 162:
oct_one = atoi("Nicaragua");
break;
case 163:
oct_one = atoi("Niger");
```

```
break;
case 164:
oct_one = atoi("Nigeria");
break;
case 165:
oct_one = atoi("Niue");
break;
case 166:
oct_one = atoi("Norfolk Island");
break;
case 167:
oct_one = atoi("Northern Mariana Islands");
break;
case 168:
oct_one = atoi("Norway");
break;
case 169:
oct_one = atoi("Oman");
break;
case 170:
oct_one = atoi("Pakistan");
break;
case 171:
oct_one = atoi("Palau");
break;
case 172:
oct_one = atoi("Palestine");
break;
case 173:
oct_one = atoi("Panama");
break;
case 174:
oct_one = atoi("Papua New Guinea");
break;
case 175:
oct_one = atoi("Paraguay");
```

```
break;
case 176:
oct_one = atoi("Peru");
break;
case 177:
oct_one = atoi("Philippines");
break;
case 178:
oct_one = atoi("Pitcairn Islands");
break;
case 179:
oct_one = atoi("Poland");
break;
case 180:
oct_one = atoi("Portugal");
break;
case 181:
oct_one = atoi("Puerto Rico");
break;
case 182:
oct_one = atoi("Qatar");
break;
case 183:
oct_one = atoi("Réunion");
break;
case 184:
oct_one = atoi("Romania");
break;
case 185:
oct_one = atoi("Rwanda");
break;
case 186:
oct_one = atoi("Russia");
break;
case 187:
oct_one = atoi("Saint Barthélemy");
```

```
break;
case 188:
oct_one = atoi("Saint Helena, Ascension and Tristan da Cunha");
break;
case 189:
oct_one = atoi("Saint Kitts and Nevis");
break;
case 190:
oct_one = atoi("Saint Lucia");
break;
case 191:
oct_one = atoi("Saint Martin");
break;
case 192:
oct_one = atoi("Saint Pierre and Miquelon");
break;
case 193:
oct_one = atoi("Saint Vincent and the Grenadines");
break;
case 194:
oct_one = atoi("Samoa");
break;
case 195:
oct_one = atoi("San Marino");
break;
case 196:
oct_one = atoi("Sao Tome and Principe");
break;
case 197:
oct_one = atoi("Saudi Arabia");
break;
case 198:
oct_one = atoi("Senegal");
break;
case 199:
oct_one = atoi("Serbia");
```

```
break;
case 200:
oct_one = atoi("Seychelles");
break;
case 201:
oct_one = atoi("Sierra Leone");
break;
case 202:
oct_one = atoi("Sint Maarten");
break;
case 203:
oct_one = atoi("Singapore");
break;
case 204:
oct_one = atoi("Slovakia");
break;
case 205:
oct_one = atoi("Slovenia");
break;
case 206:
oct_one = atoi("Solomon Islands");
break;
case 207:
oct_one = atoi("Somalia");
break;
case 208:
oct_one = atoi("South Africa");
break;
case 209:
oct_one = atoi("South Georgia and the South Sandwich Islands");
break;
case 210:
oct_one = atoi("Spain");
break;
case 211:
oct_one = atoi("Sri Lanka");
```

```
break;
case 212:
oct_one = atoi("Sudan");
break;
case 213:
oct_one = atoi("Suriname");
break;
case 214:
oct_one = atoi("Swaziland");
break;
case 215:
oct_one = atoi("Sweden");
break;
case 216:
oct_one = atoi("Switzerland");
break;
case 217:
oct_one = atoi("Svalbard and Jan Mayen");
break;
case 218:
oct_one = atoi("Syria");
break;
case 219:
oct_one = atoi("Taiwan");
break;
case 220:
oct_one = atoi("Tajikistan");
break;
case 221:
oct_one = atoi("Tanzania");
break;
case 222:
oct_one = atoi("Thailand");
break;
case 223:
oct_one = atoi("Togo");
```

```
break;
case 224:
oct_one = atoi("Tokelau");
break;
case 225:
oct_one = atoi("Tonga");
break;
case 226:
oct_one = atoi("Trinidad and Tobago");
break;
case 227:
oct_one = atoi("Tunisia");
break;
case 228:
oct_one = atoi("Turkey");
break;
case 229:
oct_one = atoi("Turkmenistan");
break;
case 230:
oct_one = atoi("Turks and Caicos Islands");
break;
case 231:
oct_one = atoi("Tuvalu");
break;
case 232:
oct_one = atoi("Uganda");
break;
case 233:
oct_one = atoi("Ukraine");
break;
case 234:
oct_one = atoi("United Arab Emirates");
break;
case 235:
oct_one = atoi("United Kingdom");
```

```
break;
case 236:
oct_one = atoi("United States");
break;
case 237:
oct_one = atoi("Uruguay");
break;
case 238:
oct_one = atoi("U.S. Virgin Islands");
break;
case 239:
oct_one = atoi("Uzbekistan");
break;
case 240:
oct_one = atoi("Vanuatu");
break;
case 241:
oct_one = atoi("Vatican");
break;
case 242:
oct_one = atoi("Venezuela");
break;
case 243:
oct_one = atoi("Vietnam");
break;
case 244:
oct_one = atoi("Wallis and Futuna");
break;
case 245:
oct_one = atoi("Yemen");
break;
case 246:
oct_one = atoi("Zambia");
break;
case 247:
oct_one = atoi("Zimbabwe");
```

```
break;
default:
break;
}
#define IN_CLASSB_HOST ((in_addr_t)) & 0x00ff0000) = oct_two
/*#define IN_CLASSB_HOST oct_two */
switch(oct_two)
{
case 1:
oct_two = atoi("Aerohive Networks");
break;
case 2:
oct_two = atoi("Alaxala Networks");
break;
case 3:
oct_two = atoi("Nokia");
break;
case 4:
oct_two = atoi("Allied Telesis");
break;
case 5:
oct_two = atoi("Asus");
break;
case 6:
oct_two = atoi("Avaya");
break;
case 7:
oct_two = atoi("AVM");
break;
case 8:
oct_two = atoi("Brocade");
break;
case 9:
oct_two = atoi("Billion Electric");
break;
case 10:
```

```
oct_two = atoi("Buffalo Technology");
break;
case 11:
oct_two = atoi("Cisco Systems");
break;
case 12:
oct_two = atoi("Dell");
break;
case 13:
oct_two = atoi("D-Link");
break;
case 14:
oct_two = atoi("Digisol");
break;
case 15:
oct_two = atoi("Enterasys");
break;
case 16:
oct_two = atoi("Ericsson");
break;
case 17:
oct_two = atoi("Extreme Networks");
break;
case 18:
oct_two = atoi("HPE");
break;
case 19:
oct_two = atoi("Huawei Routers");
break;
case 20:
oct_two = atoi("Juniper Networks");
break;
case 21:
oct_two = atoi("LANCOM Systems");
break;
case 22:
```

```
oct_two = atoi("Linksys");
break;
case 23:
oct_two = atoi("Meraki");
break;
case 24:
oct_two = atoi("Mitsubishi");
break;
case 25:
oct_two = atoi("NEC");
break;
case 26:
oct_two = atoi("Netgear");
break;
case 27:
oct_two = atoi("Nokia Networks");
break;
case 28:
oct_two = atoi("RAD Data Communications");
break;
case 29:
oct_two = atoi("Ruckus");
break;
case 30:
oct_two = atoi("Sierra Wireless");
break;
case 31:
oct_two = atoi("Telco Systems");
break;
case 32:
oct_two = atoi("TP-Link");
break;
case 33:
oct_two = atoi("Ubiquiti");
break;
case 34:
```

```
oct_two = atoi("Xirrus");
break;
case 35:
oct_two = atoi("Yamaha");
break;
case 36:
oct_two = atoi("ZyXEL");
break;
case 37:
oct_two = atoi("ZTE");
break;
default:
break;
}
#define IN_CLASSC_HOST ((in_addr_t) & 0x0000ff00) = oct_three
/*#define IN_CLASSC_NET 0xffffff00
#define IN_CLASSC_NSHIFT 8
#define IN_CLASSC_HOST oct_three */
#define IN_CLASSD_HOST ((in_addr_t) & 0x000000ff)= oct_four
/* #define IN_CLASSD_HOST oct_four */
#endif
}
#define INET_ADDRSTRLEN 16
#define INET6_ADDRSTRLEN 46
/* Structure describing an Internet socket address */
struct sockaddr_in
{
__SOCKADDR_COMMON (sin_);
in_port_t sin_port; /* Port number.  */
struct in_addr sin_addr; /* Internet address.*/
 unsigned char sin_zero[sizeof (struct sockaddr) -
__SOCKADDR_COMMON_SIZE -
sizeof (in_port_t) -
sizeof (struct in_addr)];
};
#ifdef __USE_MISC
```

```
/* IPv4 multicast request.  */
struct ip_mreq
{
/* IP multicast address of group.  */
struct in_addr imr_multiaddr;
/* Local IP address of interface.  */
 struct in_addr imr_interface;
};
struct ip_mreq_source
{
/* IP multicast address of group.  */
struct in_addr imr_multiaddr;
/* IP address of source.  */
struct in_addr imr_interface;
/* IP address of interface.  */
struct in_addr imr_sourceaddr;
};
#endif
#ifdef __USE_MISC
/* Multicast group request.  */
struct group_req
{
/* Interface index.  */
uint32_t gr_interface;
/* Group address.  */
struct sockaddr_storage gr_group;
};
struct group_source_req
{
/* Interface index.  */
uint32_t gsr_interface;
/* Group address.  */
struct sockaddr_storage gsr_group;
/* Source address.  */
struct sockaddr_storage gsr_source;
};
```

```
/* Full-state filter operations.  */
struct ip_msfilter
  {
/* IP multicast address of group.  */
struct in_addr imsf_multiaddr;
/* Local IP address of interface.  */
struct in_addr imsf_interface;
/* Filter mode.  */
uint32_t imsf_fmode;
/* Number of source addresses.  */
uint32_t imsf_numsrc;
/* Source addresses.  */
struct in_addr imsf_slist[1];
};
#define IP_MSFILTER_SIZE(numsrc) (sizeof (struct ip_msfilter) \
- sizeof (struct in_addr)\
+ (numsrc) * sizeof (struct in_addr))

struct group_filter
{
/* Interface index.  */
uint32_t gf_interface;
/* Group address.  */
struct sockaddr_storage gf_group;
/* Filter mode.  */
uint32_t gf_fmode;
/* Number of source addresses.  */
uint32_t gf_numsrc;
/* Source addresses.  */
struct sockaddr_storage gf_slist[1];
};
#define GROUP_FILTER_SIZE(numsrc) (sizeof (struct group_filter) \
- sizeof (struct sockaddr_storage\
+ ((numsrc\
* sizeof (struct sockaddr_storage)))
#endif
```

```
extern uint32_t ntohl (uint32_t __netlong) __THROW __attribute__ ((__const__));
extern uint16_t ntohs (uint16_t __netshort)
__THROW __attribute__ ((__const__));
extern uint32_t htonl (uint32_t __hostlong)
__THROW __attribute__ ((__const__));
extern uint16_t htons (uint16_t __hostshort)
__THROW __attribute__ ((__const__));
#include <endian.h>
/* Get machine dependent optimized versions of byte swapping functions.  */
#include <bits/byteswap.h>
#ifdef __OPTIMIZE__
/* We can optimize calls to the conversion functions.  Either nothing has
to be done or we are using directly the byte-swapping functions which
often can be inlined.  */
# if __BYTE_ORDER == __BIG_ENDIAN
/* The host byte order is the same as network byte order,
so these functions are all just identity*/
# define ntohl(x) (x)
# define ntohs(x) (x)
# define htonl(x) (x)
# define htons(x) (x)
# else
#  if __BYTE_ORDER == __LITTLE_ENDIAN
#define ntohl(x) __bswap_32 (x)
#define ntohs(x) __bswap_16 (x)
#define htonl(x) __bswap_32 (x)
#define htons(x) __bswap_16 (x)
#  endif
# endif
#endif
#ifdef __GNUC__
#ifdef __USE_GNU
struct cmsghdr; /* Forward declaration.  */
/* Multicast source filter support.  */
/* Get IPv4 source filter.  */
extern int getipv4sourcefilter (int __s, struct in_addr __interface_addr,
```

```
    struct in_addr __group, uint32_t *__fmode,
    uint32_t *__numsrc, struct in_addr *__slist)
    __THROW;
    /* Set IPv4 source filter.  */
    extern int setipv4sourcefilter (int __s, struct in_addr __interface_addr,
    struct in_addr __group, uint32_t __fmode,
    uint32_t __numsrc,
    const struct in_addr *__slist)
    __THROW;
    /* Get source filter.  */
    extern int getsourcefilter (int __s, uint32_t __interface_addr,
    const struct sockaddr *__group,
    socklen_t __grouplen, uint32_t *__fmode,
    uint32_t *__numsrc,
    struct sockaddr_storage *__slist) __THROW;
    /* Set source filter.  */
    extern int setsourcefilter (int __s, uint32_t __interface_addr,
    const struct sockaddr *__group,
    socklen_t __grouplen, uint32_t __fmode,
    uint32_t __numsrc,
    const struct sockaddr_storage *__slist) __THROW;
    #endif /* use GNU */
    __END_DECLS
#endif /* netinet/in2.h
```

Appendix B

Receiver main Program

```
#include <arpa/inet1.h>
#include <linux/if_packet.h>
#include <linux/udp.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/ether.h>
#include <netinet/ip3.h>
#include <unistd.h>
#define DEST_MAC0 0x08
#define DEST_MAC1 0x00
#define DEST_MAC2 0x27
#define DEST_MAC3 0x95
#define DEST_MAC4 0x69
#define DEST_MAC5 0x1c
#define ETHER_TYPE 0xBAD0
#define DEFAULT_IF "eth0"
#define BUF_SIZ 1024
#define SIZE 32
int main(int argc, char *argv[])
{
char sender[INET6_ADDRSTRLEN];
int sockfd, ret,i,x=0;
int sockopt;
ssize_t numbytes;
struct ifreq ifopts; /* set promiscuous mode */
struct ifreq if_ip; /* get ip addr */
struct sockaddr_storage their_addr;
uint8_t buf[BUF_SIZ];
```

```
char ifName[IFNAMSIZ];
char output[SIZE];
/* Get interface name */
if (argc > 1)
strcpy(ifName, argv[1]);
else
strcpy(ifName, DEFAULT_IF);
/* Header structures */
struct ether_header *eh = (struct ether_header *) buf;
struct iphdr *iph = (struct iphdr *) (buf + sizeof(struct ether_header));
struct udphdr *udph = (struct udphdr *) (buf + sizeof(struct iphdr) + sizeof(struct
ether_header));
memset(&if_ip, 0, sizeof(struct ifreq));
/* Open PF_PACKET socket, listening for Ethertype ETHER_TYPE */
if ((sockfd = socket(PF_PACKET, SOCK_RAW, htons(ETHER_TYPE))) == -1) {
perror("listener: socket");
return -1;
}
/* Set interface to promiscuous mode - do we need to do this every time? */
strncpy(ifopts.ifr_name, ifName, IFNAMSIZ-1);
ioctl(sockfd, SIOCGIFFLAGS, &ifopts);
ifopts.ifr_flags |= IFF_PROMISC;
ioctl(sockfd, SIOCSIFFLAGS, &ifopts);
/* Allow the socket to be reused - incase connection is closed prematurely */
if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &sockopt, sizeof sockopt) == -1) {
perror("setsockopt");
close(sockfd);
exit(EXIT_FAILURE);
}
/* Bind to device */
if (setsockopt(sockfd, SOL_SOCKET, SO_BINDTODEVICE, ifName, IFNAMSIZ-1) == -1) {
perror("SO_BINDTODEVICE");
close(sockfd);
exit(EXIT_FAILURE);
}
repeat:
numbytes = recvfrom(sockfd, buf, BUF_SIZ, 0, NULL, NULL);
```

```
/* Check the packet is for me */
if (eh->ether_dhost[0] == DEST_MAC0 &&
eh->ether_dhost[1] == DEST_MAC1 &&
eh->ether_dhost[2] == DEST_MAC2 &&
eh->ether_dhost[3] == DEST_MAC3 &&
eh->ether_dhost[4] == DEST_MAC4 &&
eh->ether_dhost[5] == DEST_MAC5) {
}
else {
ret = -1;
goto done;
}
((struct sockaddr_in *)&their_addr)->sin_addr.s_addr = iph->saddr;
inet_ntop(AF_INET, &((struct sockaddr_in*)&their_addr)->sin_addr, sender, sizeof
sender);
if(strcmp(sender,"192.168.1.78")==0)
{
strncpy(if_ip.ifr_name, ifName, IFNAMSIZ-1);
if (ioctl(sockfd, SIOCGIFADDR, &if_ip) >= 0) { /* if we can't check then don't */
/*printf("Source IP: %s\n My IP: %s\n", sender,
inet_ntoa(((struct sockaddr_in *)&if_ip.ifr_addr)->sin_addr));*/
/* ignore if I sent it */
if (strcmp(sender, inet_ntoa(((struct sockaddr_in *)&if_ip.ifr_addr)->sin_addr)) ==
0) {
printf("but I sent it :(\n");
ret = -1;
goto done;
}
}
printf("Source IP: %s\n My IP: %s\n", sender,
inet_ntoa(((struct sockaddr_in *)&if_ip.ifr_addr)->sin_addr));
ret = ntohs(udph->len) - sizeof(struct udphdr);
/* Print packet */
if(buf[46] == 0x48)
{
printf("Unencrypted Data: ");
for (i=46; i<numbytes-2; i++)
```

```
{
output[x] = (char)buf[i];
if(numbytes==0)break;
x++;
}
printf("%s\n",output);
x=0;
printf("Public Key = %x\n",(iph->opt));
}
else
{
printf("Encrypted Data: ");
for (i=46; i<numbytes-3; i++)
{
output[x] = (char)buf[i];
if(numbytes==0)break;
x++;
}
printf("%s\n",output);
x=0;
printf("Public Key = %x\n",(iph->opt));
}
printf("\n\n");
done: goto repeat;
close(sockfd);
return ret;
}
else
{
goto repeat;
close(sockfd);
return ret;
}
}
```

=> Receiver in1.h and ip3.h header file remains same as Sender in1.h and ip3.h files with the exception that in Receiver ip3.h header file value of custom IP_Option field is not defined as Option field values as they sent by the Sender.

References

Sites

1) https://www.youtube.com/watch?v=EE--H1wYKGU
2) https://www.bluetin.io/pi-wars-2018/beginners-guide-raspberry-pi-configuration/
3) https://www.avoiderrors.com/install-raspbian-virtualbox/
4) http://www.circuitbasics.com/how-to-set-up-a-static-ip-on-the-raspberry-pi/
5) https://www.youtube.com/watch?v=1Sx_qtF_lNk
6) http://www.ronnutter.com/raspberry-pi-intro-to-tcpdump/
7) https://raspberrypi.stackexchange.com/questions/71399/how-to-edit-config-txt
8) https://www.raspberrypi.org/forums/viewtopic.php?t=73393
9) https://www.youtube.com/watch?v=FV7eiqN01hc
10) https://www.raspberrypi.org/forums/viewtopic.php?t=7429
11) https://www.raspberrypi.org/forums/viewtopic.php?t=168071
12) https://austinmarton.wordpress.com/2011/09/14/sending-raw-ethernet-packets-from-a-specific-interface-in-c-on-linux/
13) https://gist.github.com/austinmarton/2862515
14) https://www.linuxbabe.com/debian/install-virtualbox-guest-additions-debian-9-stretch
15) https://www.browserling.com/tools/text-to-hex
16) https://www.engineersgarage.com/c-language-programs/display-month-name-according-month-number-using-switch-statement
17) https://www.includehelp.com/c-programs/format-ip-address-octets.aspx
18) https://stackoverflow.com/questions/6530578/what-are-addresses-of-type-in-addr-t-inet-ntoa-etc#6531669
19) https://stackoverflow.com/questions/20778903/getting-warning-in-c-for-atoi-function
20) https://en.wikipedia.org/wiki/List_of_postal_codes
21) https://cryptii.com/pipes/integer-encoder
22) https://gist.github.com/skandhas/8839282/revisions
23) https://tools.ietf.org/html/rfc791#page-15
24) https://www.thegeekstuff.com/2011/12/c-socket-programming/?utm_source=feedburner

25) https://codereview.stackexchange.com/questions/13461/two-way-communication-in-tcp-server-client-implementation
26) http://man7.org/linux/man-pages/man7/packet.7.html
27) https://c-for-dummies.com/blog/?p=2252
28) https://stackoverflow.com/questions/8060170/printing-hexadecimal-characters-in-c
29) https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/
30) https://www.tutorialspoint.com/cprogramming/c_structures.htm
31) https://fresh2refresh.com/c-programming/c-pointer/
32) https://opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/
33) https://stackoverflow.com/questions/3649026/how-to-display-hexadecimal-numbers-in-c
34) https://www.includehelp.com/c-programs/extract-bytes-from-int.aspx
35) https://tools.ietf.org/html/rfc7126
36) https://lb.raspberrypi.org/forums/viewtopic.php?t=40294
37) https://gist.github.com/skandhas
38) https://codeforwin.org/2015/09/c-program-to-convert-hexadecimal-to-decimal-number-system.html
39) https://www.geeksforgeeks.org/program-decimal-hexadecimal-conversion/
40) https://www.quora.com/Whats-the-difference-between-the-AF_PACKET-and-AF_INET-in-python-socket
41) https://itsfoss.com/find-which-kernel-version-is-running-in-ubuntu/
42) https://stackoverflow.com/questions/24590818/what-is-the-difference-between-ipproto-ip-and-ipproto-raw
43) https://en.wikipedia.org/wiki/List_of_networking_hardware_vendors
44) https://en.wikipedia.org/wiki/Caesar_cipher
45) https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
46) http://codingloverlavi.blogspot.com/2015/04/diffie-hellman-key-exchange-algorithm.html