Capturing Ethernet Frames using in C with Winpcap API

Hakan Everdi 2017555022

Ege Boz 2020555459

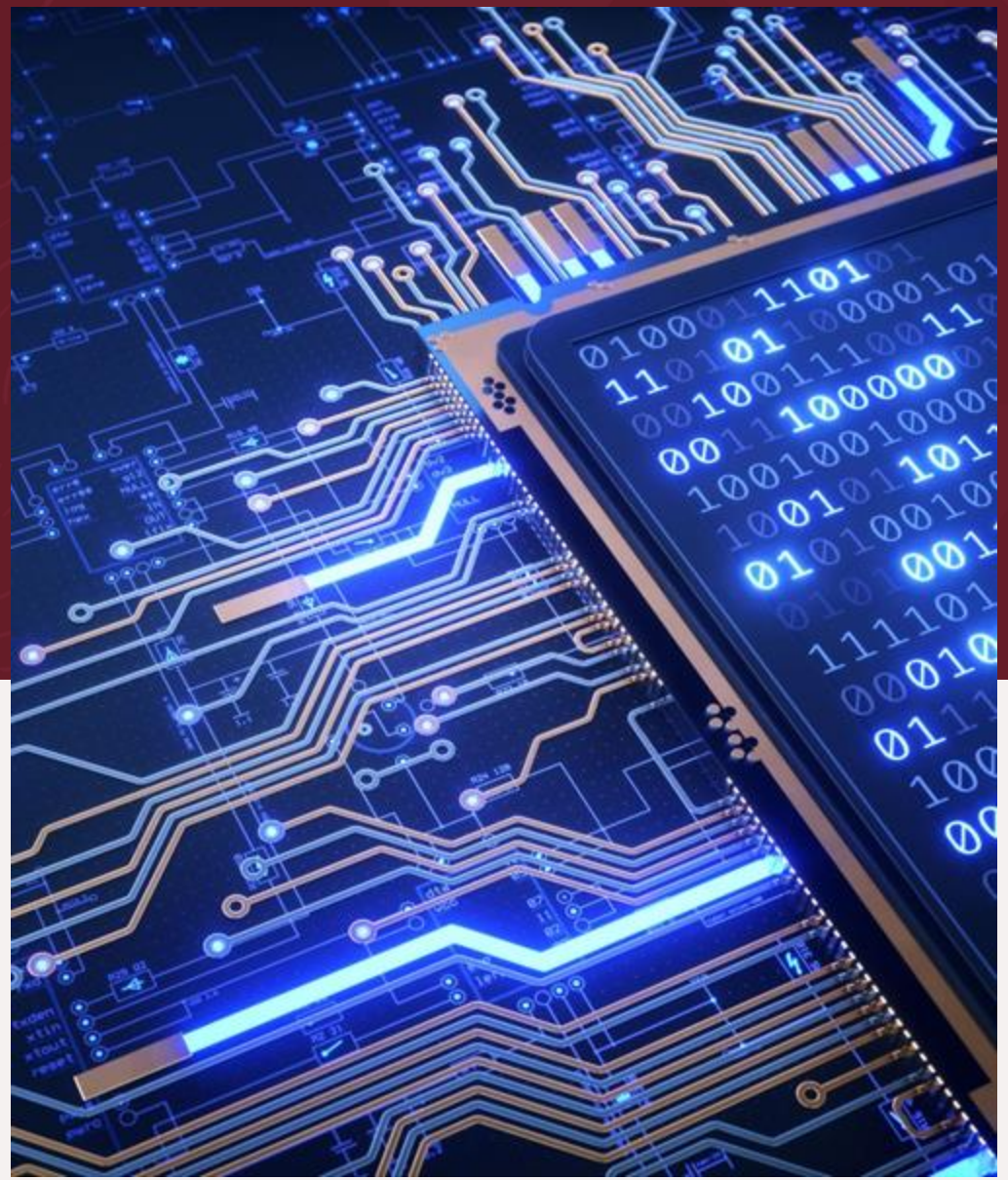
Ahmet Yalavaç 2018555067

AGENDA

- Information about WinPcap
- Basic libpcap program
  - Obtaining the device list
  - Simple Sniffer with winpcap
  - Packet structure

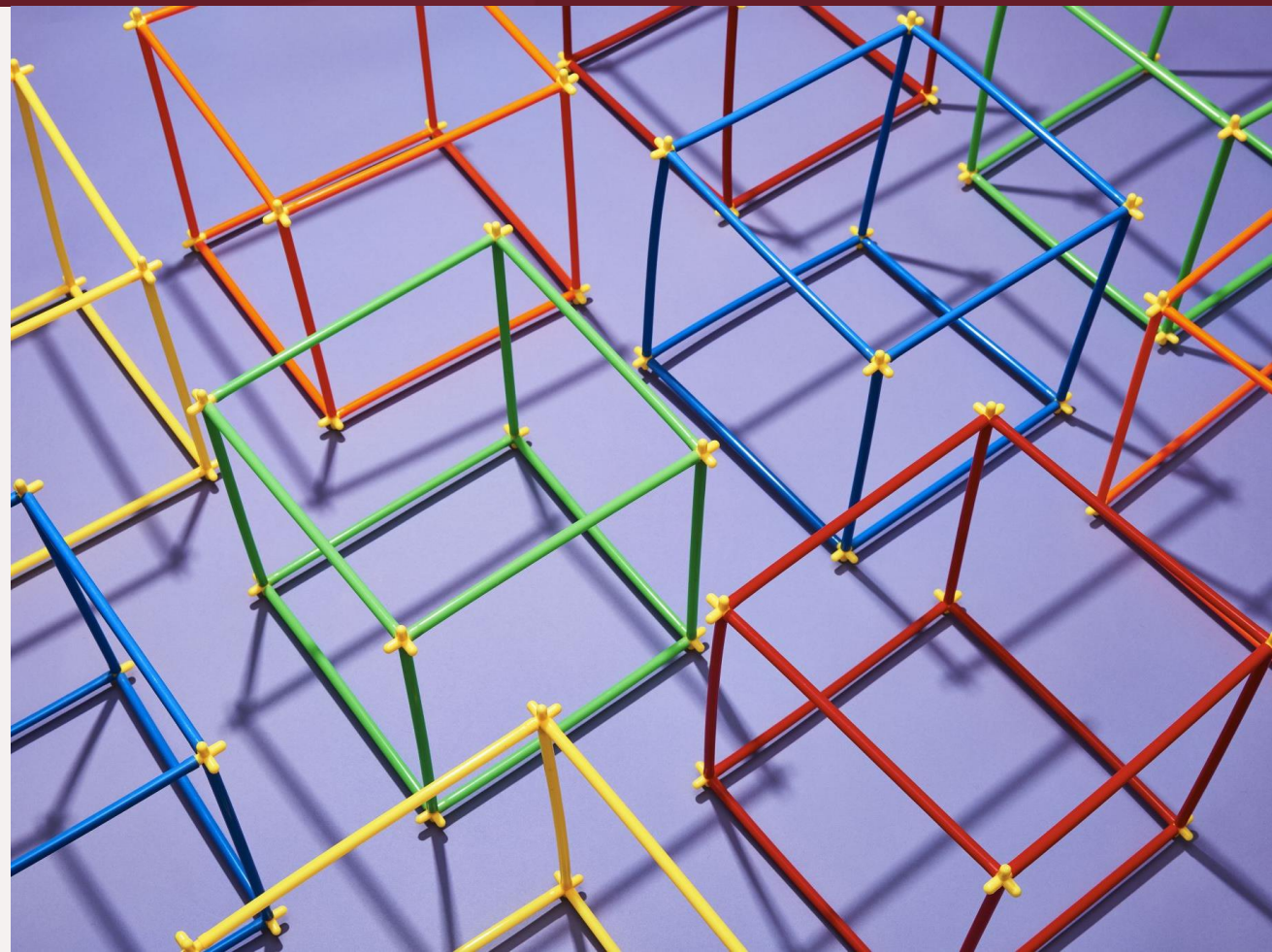
What is WinPcap?

WinPcap is a packet capture library for Windows used for packet sniffing and sending raw packets.



What is the use of the WinPcap library?

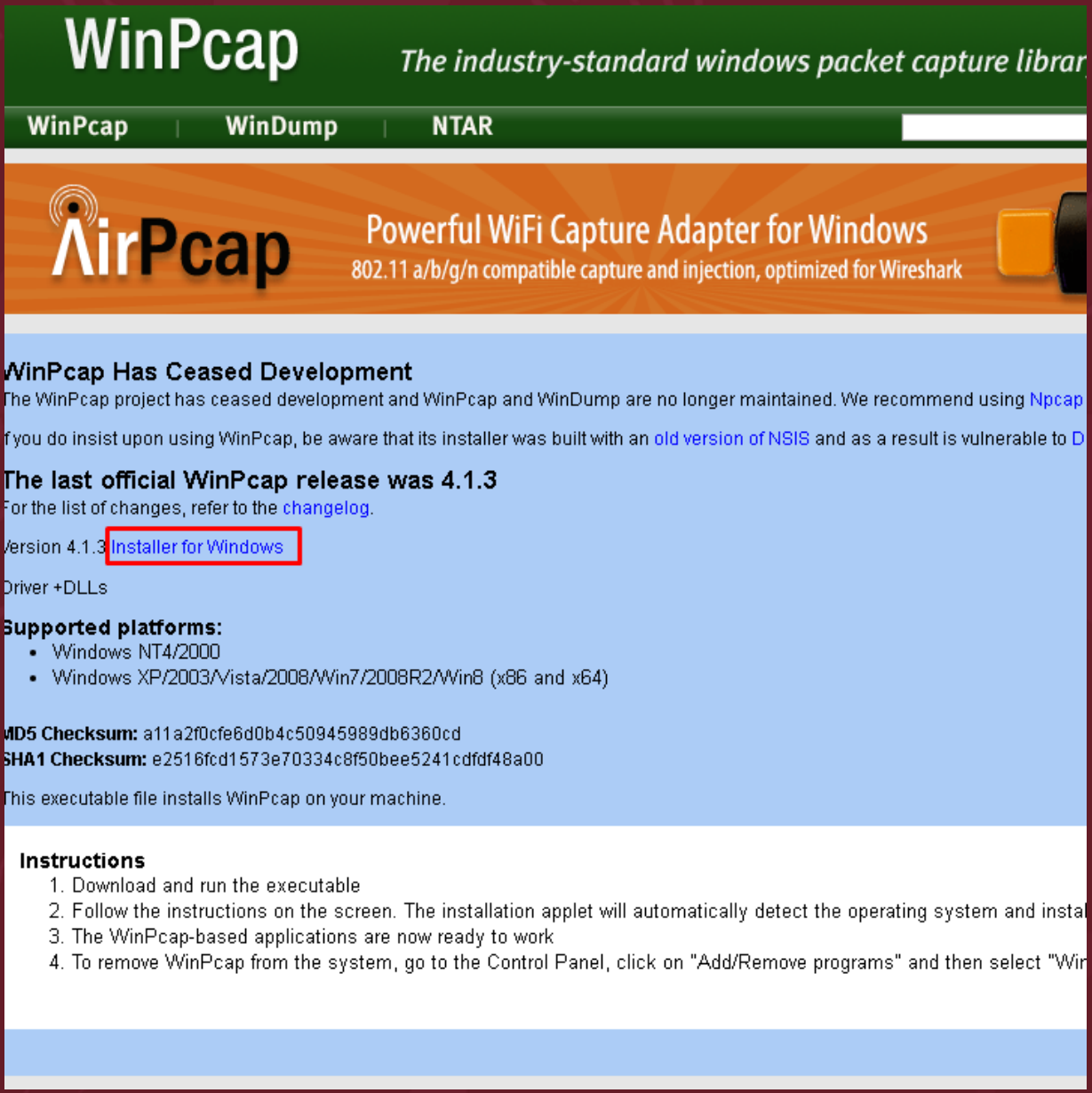
- WinPcap is the industry-standard tool for link-layer network access in Windows environments: it allows applications to capture and transmit network packets bypassing the protocol stack, and has additional useful features, including kernel-level packet filtering, a network statistics engine and support for remote packet capture. WinPcap consists of a driver, that extends the operating system to provide low-level network access, and a library that is used to easily access the low-level network layers. This library also contains the Windows version of the well known libpcap Unix API.

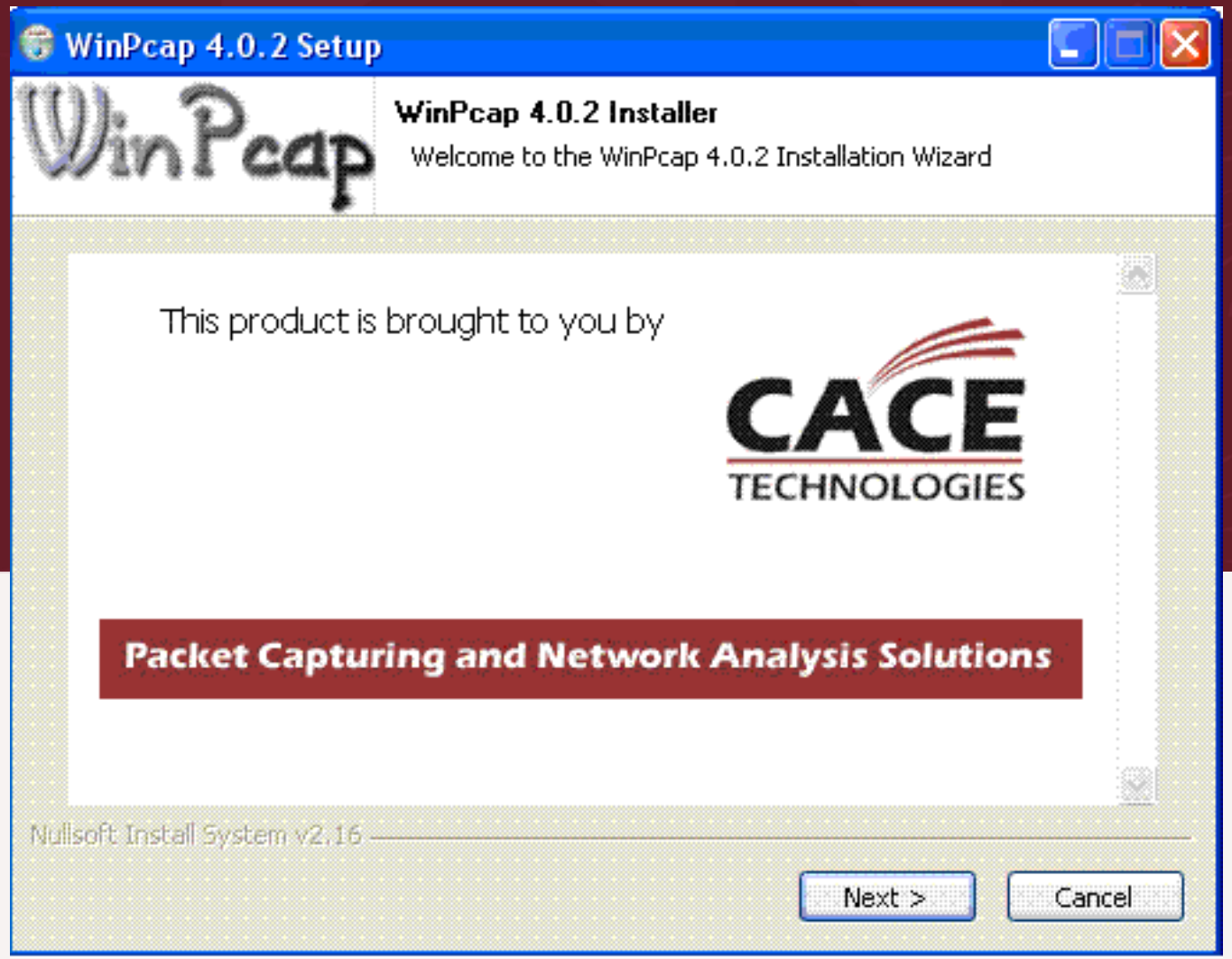


How to Download WinPcap for Windows

- WinPcap library is provided by the Wireshark developers from the winpcap.org web site. From the following URL, we can land to the download page:

https://www.winpcap.org/install/default.htm

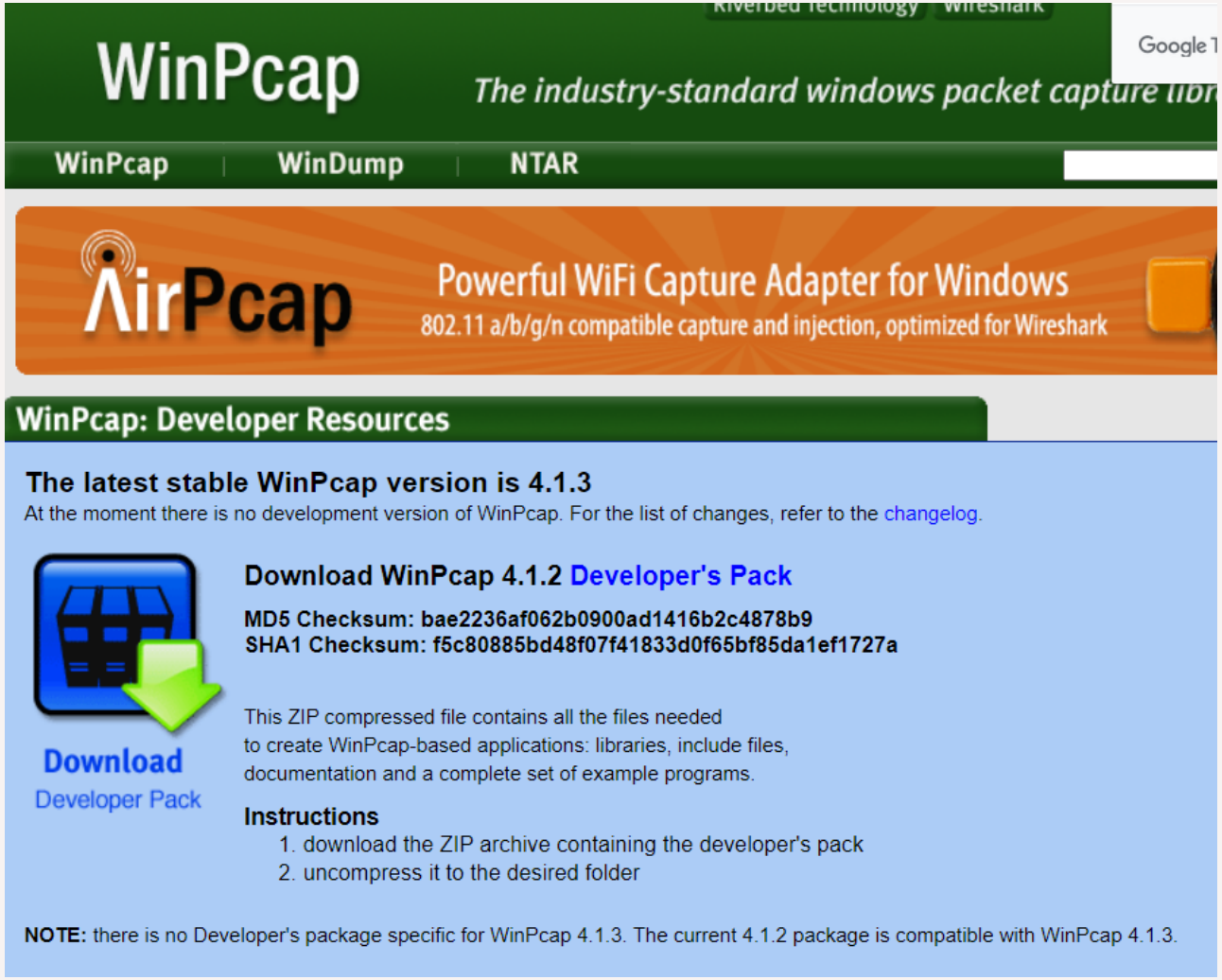




After downloading and running the executable file, following the instructions given by the wizard, the WinPcap components are installed on your computer.

The next step is to download the WinPcap developer's pack and link to an IDE. This can be obtained from the following location:

http://www.winpcap.org/devel.htm



Now, we are finally ready to start using WinPcap's power to develop strong, networking applications. I will show you examples.

Obtaining the device list

- Typically, the first thing that a WinPcap-based application does is get a list of attached network adapters. Both libpcap and WinPcap provide the **pcap_findalldevs_ex()** function for this purpose: this function returns a linked list of **pcap_if** structures, each of which contains comprehensive information about an attached adapter. In particular, the fields *name* and *description* contain the name and a human readable description, respectively, of the corresponding device.

```c
#include "pcap.h"

main()
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int i=0;
    char errbuf[PCAP_ERRBUF_SIZE];

    /* Retrieve the device list from the local machine */
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL /* auth is not needed */, &alldevs, errbuf) == -1)
    {
        fprintf(stderr,"Error in pcap_findalldevs_ex: %s\n", errbuf);
        exit(1);
    }

    /* Print the list */
    for(d= alldevs; d != NULL; d= d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)\n", d->description);
        else
            printf(" (No description available)\n");
    }

    if (i == 0)
    {
        printf("\nNo interfaces found! Make sure WinPcap is installed.\n");
        return;
    }

    /* We don't need any more the device list. Free it */
    pcap_freealldevs(alldevs);
}
```

First of all, pcap_findalldevs_ex(), like other libpcap functions, has an *errbuf* parameter.
This parameter points to a string filled by libpcap with a description of the error if something goes wrong.
Note that we free the list with pcap_freealldevs() once when we have finished with it.

Opening an adapter and capturing the packets

- Now that we've seen how to obtain an adapter to play with, let's start the real job, opening an adapter and capturing some traffic. Now i will show you a program that prints some information about each packet flowing through the adapter.

```c
#include "pcap.h"

/* prototype of the packet handler */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data);

int main()
{
pcap_if_t *alldevs;
pcap_if_t *d;
int inum;
int i=0;
pcap_t *adhandle;
char errbuf[PCAP_ERRBUF_SIZE];

    /* Retrieve the device list on the local machine */
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
    {
        fprintf(stderr,"Error in pcap_findalldevs: %s\n", errbuf);
        exit(1);
    }

    /* Print the list */
    for(d=alldevs; d; d=d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)\n", d->description);
        else
            printf(" (No description available)\n");
    }

    if(i==0)
    {
        printf("\nNo interfaces found! Make sure WinPcap is installed.\n");
        return -1;
    }

    printf("Enter the interface number (1-%d):",i);
    scanf_s("%d", &inum);

    if(inum < 1 || inum > i)
    {
        printf("\nInterface number out of range.\n");
        /* Free the device list */
        pcap_freealldevs(alldevs);
        return -1;
    }
```

```c
    /* Jump to the selected adapter */
    for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);

    /* Open the device */
    if ( (adhandle= pcap_open(d->name,          // name of the device
                              65536,            // portion of the packet to capture
                                                // 65536 guarantees that the whole packet will be captured on all the link layers
                              PCAP_OPENFLAG_PROMISCUOUS,    // promiscuous mode
                              1000,             // read timeout
                              NULL,             // authentication on the remote machine
                              errbuf            // error buffer
                              ) ) == NULL)
    {
        fprintf(stderr,"\nUnable to open the adapter. %s is not supported by WinPcap\n", d->name);
        /* Free the device list */
        pcap_freealldevs(alldevs);
        return -1;
    }

    printf("\nlistening on %s...\n", d->description);

    /* At this point, we don't need any more the device list. Free it */
    pcap_freealldevs(alldevs);

    /* start the capture */
    pcap_loop(adhandle, 0, packet_handler, NULL);

    return 0;
}


/* Callback function invoked by libpcap for every incoming packet */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data)
{
    struct tm ltime;
    char timestr[16];
    time_t local_tv_sec;

    /*
     * unused variables
     */
    (VOID)(param);
    (VOID)(pkt_data);

    /* convert the timestamp to readable format */
    local_tv_sec = header->ts.tv_sec;
    localtime_s(&ltime, &local_tv_sec);
    strftime( timestr, sizeof timestr, "%H:%M:%S", &ltime);

    printf("%s,%.6d len:%d\n", timestr, header->ts.tv_usec, header->len);

}
```

The function that opens a capture device is pcap_open(). The parameters, snaplen, flags and to_ms deserve some explanation.

Snaplen, specifies the portion of the packet to capture. On some OSes (like xBSD and Win32), the packet driver can be configured to capture only the initial part of any packet: this decreases the amount of data to copy to the application and therefore improves the efficiency of the capture. In this case we use the value 65536 which is higher than the greatest MTU that we could encounter. In this manner we ensure that the application will always receive the whole packet.

flags: the most important flag is the one that indicates if the adapter will be put in promiscuous mode. In normal operation, an adapter only captures packets from the network that are destined to it; the packets exchanged by other hosts are therefore ignored. Instead, when the adapter is in promiscuous mode it captures all packets whether they are destined to it or not. This means that on shared media (like non-switched Ethernet), WinPcap will be able to capture the packets of other hosts. Promiscuous mode is the default for most capture applications, so we enable it in the example.

to_ms: specifies the read timeout, in milliseconds. A read on the adapter (for example, with pcap_dispatch() or pcap_next_ex()) will always return after to_ms milliseconds, even if no packets are available from the network. to_ms also defines the interval between statistical reports if the adapter is in statistical mode. Setting to_ms to 0 means no timeout, a read on the adapter never returns if no packets arrive. A -1 timeout on the other side causes a read on the adapter to always return immediately.

Simple Sniffer with winpcap , prints ethernet , ip , tcp , udp and icmp headers along with data dump in hex

```c
#include "stdio.h"
#include "winsock2.h"   //need winsock for inet_ntoa and ntohs methods

#define HAVE_REMOTE
#include "pcap.h"       //Winpcap :)

#pragma comment(lib , "ws2_32.lib") //For winsock
#pragma comment(lib , "wpcap.lib") //For winpcap

//some packet processing functions
void ProcessPacket (u_char* , int); //This will decide how to digest

void print_ethernet_header (u_char*);
void PrintIpHeader (u_char* , int);
void PrintIcmpPacket (u_char* , int);
void print_udp_packet (u_char* , int);
void PrintTcpPacket (u_char* , int);
void PrintData (u_char* , int);

// Set the packing to a 1 byte boundary
//#include "pshpack1.h"
//Ethernet Header
typedef struct ethernet_header
{
        UCHAR dest[6];
        UCHAR source[6];
        USHORT type;
}   ETHER_HDR , *PETHER_HDR , FAR * LPETHER_HDR , ETHERHeader;

//Ip header (v4)
typedef struct ip_hdr
{
        unsigned char ip_header_len:4; // 4-bit header length (in 32-bit words) normally=5 (Means 20 Bytes may be 24 also)
        unsigned char ip_version :4; // 4-bit IPv4 version
        unsigned char ip_tos; // IP type of service
        unsigned short ip_total_length; // Total length
        unsigned short ip_id; // Unique identifier

        unsigned char ip_frag_offset :5; // Fragment offset field

        unsigned char ip_more_fragment :1;
        unsigned char ip_dont_fragment :1;
        unsigned char ip_reserved_zero :1;

        unsigned char ip_frag_offset1; //fragment offset

        unsigned char ip_ttl; // Time to live
        unsigned char ip_protocol; // Protocol(TCP,UDP etc)
        unsigned short ip_checksum; // IP checksum
        unsigned int ip_srcaddr; // Source address
        unsigned int ip_destaddr; // Source address
} IPV4_HDR;
```

```c
//UDP header
typedef struct udp_hdr
{
        unsigned short source_port; // Source port no.
        unsigned short dest_port; // Dest. port no.
        unsigned short udp_length; // Udp packet length
        unsigned short udp_checksum; // Udp checksum (optional)
} UDP_HDR;

// TCP header
typedef struct tcp_header
{
        unsigned short source_port; // source port
        unsigned short dest_port; // destination port
        unsigned int sequence; // sequence number - 32 bits
        unsigned int acknowledge; // acknowledgement number - 32 bits

        unsigned char ns :1; //Nonce Sum Flag Added in RFC 3540.
        unsigned char reserved_part1:3; //according to rfc
        unsigned char data_offset:4; /*The number of 32-bit words in the TCP header.
        This indicates where the data begins.
        The length of the TCP header is always a multiple
        of 32 bits.*/

        unsigned char fin :1; //Finish Flag
        unsigned char syn :1; //Synchronise Flag
        unsigned char rst :1; //Reset Flag
        unsigned char psh :1; //Push Flag
        unsigned char ack :1; //Acknowledgement Flag
        unsigned char urg :1; //Urgent Flag

        unsigned char ecn :1; //ECN-Echo Flag
        unsigned char cwr :1; //Congestion Window Reduced Flag

        ////////////////////////////////

        unsigned short window; // window
        unsigned short checksum; // checksum
        unsigned short urgent_pointer; // urgent pointer
} TCP_HDR;

typedef struct icmp_hdr
{
        BYTE type; // ICMP Error type
        BYTE code; // Type sub code
        USHORT checksum;
        USHORT id;
        USHORT seq;
} ICMP_HDR;
// Restore the byte boundary back to the previous value
//#include <poppack.h>

FILE *logfile;
int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;
struct sockaddr_in source,dest;
char hex[2];
```

```c
//Its free!
ETHER_HDR *ethhdr;
IPV4_HDR *iphdr;
TCP_HDR *tcpheader;
UDP_HDR *udpheader;
ICMP_HDR *icmpheader;
u_char *data;

int main()
{
        u_int i, res , inum ;
        u_char errbuf[PCAP_ERRBUF_SIZE] , buffer[100];
        u_char *pkt_data;
        time_t seconds;
        struct tm tbreak;
        pcap_if_t *alldevs, *d;
        pcap_t *fp;
        struct pcap_pkthdr *header;

        fopen_s(&logfile , "log.txt" , "w");

        if(logfile == NULL)
        {
                printf("Unable to create file.");
        }

        /* The user didn't provide a packet source: Retrieve the local device list */
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
    {
        fprintf(stderr,"Error in pcap_findalldevs_ex: %s\n", errbuf);
        return -1;
    }

        i = 0;
    /* Print the list */
    for(d=alldevs; d; d=d->next)
    {
        printf("%d. %s\n    ", ++i, d->name);

        if (d->description)
                {
            printf(" (%s)\n", d->description);
                }
        else
                {
            printf(" (No description available)\n");
                }
    }

    if (i==0)
    {
        fprintf(stderr,"No interfaces found! Exiting.\n");
        return -1;
    }

        printf("Enter the interface number you would like to sniff : ");
        scanf_s("%d" , &inum);
```

```
        /* Jump to the selected adapter */
    for (d=alldevs, i=0; i< inum-1 ;d=d->next, i++);

    /* Open the device */
    if ( (fp= pcap_open(d->name,
                        100 /*snaplen*/,
                        PCAP_OPENFLAG_PROMISCUOUS /*flags*/,
                        20 /*read timeout*/,
                        NULL /* remote authentication */,
                        errbuf)
                        ) == NULL)
    {
        fprintf(stderr,"\nError opening adapter\n");
        return -1;
    }

        //read packets in a loop :)
    while((res = pcap_next_ex( fp, &header, &pkt_data)) >= 0)
    {
        if(res == 0)
                {
            // Timeout elapsed
            continue;
                }
                seconds = header->ts.tv_sec;
                localtime_s( &tbreak , &seconds);
                strftime (buffer , 80 , "%d-%b-%Y %I:%M:%S %p" , &tbreak );
        //print pkt timestamp and pkt len
        //fprintf(logfile , "\nNext Packet : %ld:%ld (Packet Length : %ld bytes) " , header->ts.tv_sec, header->ts.tv_usec, header->len);
                fprintf(logfile , "\nNext Packet : %s.%ld (Packet Length : %ld bytes) " , buffer , header->ts.tv_usec, header->len);
                ProcessPacket(pkt_data , header->caplen);
    }

        if(res == -1)
    {
        fprintf(stderr, "Error reading the packets: %s\n" , pcap_geterr(fp) );
        return -1;
    }

        return 0;
}

void ProcessPacket(u_char* Buffer, int Size)
{
        //Ethernet header
        ethhdr = (ETHER_HDR *)Buffer;
        ++total;

        //Ip packets
        if(ntohs(ethhdr->type) == 0x0800)
        {
                //ip header
                iphdr = (IPV4_HDR *)(Buffer + sizeof(ETHER_HDR));

                switch (iphdr->ip_protocol) //Check the Protocol and do accordingly...
                {
                        case 1: //ICMP Protocol
                        icmp++;
                        PrintIcmpPacket(Buffer,Size);
                        break;
```

```
                        case 2: //IGMP Protocol
                        igmp++;
                        break;

                        case 6: //TCP Protocol
                        tcp++;
                        PrintTcpPacket(Buffer,Size);
                        break;

                        case 17: //UDP Protocol
                        udp++;
                        print_udp_packet(Buffer,Size);
                        break;

                        default: //Some Other Protocol like ARP etc.
                        others++;
                        break;
                }
        }

        printf("TCP : %d UDP : %d ICMP : %d IGMP : %d Others : %d Total : %d\r" , tcp , udp , icmp , igmp , others , total);
}

/*
        Print the Ethernet header
*/
void print_ethernet_header (u_char* buffer )
{
        ETHER_HDR *eth = (ETHER_HDR *)buffer;

        fprintf(logfile,"\n");
        fprintf(logfile,"Ethernet Header\n");
        fprintf(logfile , " |-Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth->dest[0] , eth->dest[1] , eth->dest[2] , eth->dest[3] , eth->dest[4] , eth->dest[5] );
    fprintf(logfile , " |-Source Address      : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth->source[0] , eth->source[1] , eth->source[2] , eth->source[3] , eth->source[4] , eth->source[5] );
    fprintf(logfile , " |-Protocol            : 0x%.4x \n" , ntohs(eth->type) );
}

/*
        Print the IP header for IP packets
*/
void PrintIpHeader (unsigned char* Buffer, int Size)
{
        int iphdrlen = 0;

        iphdr = (IPV4_HDR *)(Buffer + sizeof(ETHER_HDR));
        iphdrlen = iphdr->ip_header_len*4;

        memset(&source, 0, sizeof(source));
        source.sin_addr.s_addr = iphdr->ip_srcaddr;

        memset(&dest, 0, sizeof(dest));
        dest.sin_addr.s_addr = iphdr->ip_destaddr;

        print_ethernet_header(Buffer);
```

```
        fprintf(logfile,"\n");
        fprintf(logfile,"IP Header\n");
        fprintf(logfile," |-IP Version : %d\n",(unsigned int)iphdr->ip_version);
        fprintf(logfile," |-IP Header Length : %d DWORDS or %d Bytes\n",(unsigned int)iphdr->ip_header_len,((unsigned int)(iphdr->ip_header_len))*4);
        fprintf(logfile," |-Type Of Service : %d\n",(unsigned int)iphdr->ip_tos);
        fprintf(logfile," |-IP Total Length : %d Bytes(Size of Packet)\n",ntohs(iphdr->ip_total_length));
        fprintf(logfile," |-Identification : %d\n",ntohs(iphdr->ip_id));
        fprintf(logfile," |-Reserved ZERO Field : %d\n",(unsigned int)iphdr->ip_reserved_zero);
        fprintf(logfile," |-Dont Fragment Field : %d\n",(unsigned int)iphdr->ip_dont_fragment);
        fprintf(logfile," |-More Fragment Field : %d\n",(unsigned int)iphdr->ip_more_fragment);
        fprintf(logfile," |-TTL : %d\n",(unsigned int)iphdr->ip_ttl);
        fprintf(logfile," |-Protocol : %d\n",(unsigned int)iphdr->ip_protocol);
        fprintf(logfile," |-Checksum : %d\n",ntohs(iphdr->ip_checksum));
        fprintf(logfile," |-Source IP : %s\n",inet_ntoa(source.sin_addr));
        fprintf(logfile," |-Destination IP : %s\n",inet_ntoa(dest.sin_addr));
}

/*
        Print the TCP header for TCP packets
*/
void PrintTcpPacket(u_char* Buffer, int Size)
{
        unsigned short iphdrlen;
        int header_size = 0 , tcphdrlen , data_size;

        iphdr = (IPV4_HDR *)(Buffer + sizeof(ETHER_HDR));
        iphdrlen = iphdr->ip_header_len*4;

        tcpheader = (TCP_HDR*)( Buffer + iphdrlen + sizeof(ETHER_HDR) );
        tcphdrlen = tcpheader->data_offset*4;

        data = ( Buffer + sizeof(ETHER_HDR) + iphdrlen + tcphdrlen );
        data_size = (Size - sizeof(ETHER_HDR) - iphdrlen - tcphdrlen );

        fprintf(logfile,"\n\n**********************TCP Packet************************\n");

        PrintIpHeader(Buffer,Size);

        fprintf(logfile,"\n");
        fprintf(logfile,"TCP Header\n");
        fprintf(logfile," |-Source Port : %u\n",ntohs(tcpheader->source_port));
        fprintf(logfile," |-Destination Port : %u\n",ntohs(tcpheader->dest_port));
        fprintf(logfile," |-Sequence Number : %u\n",ntohl(tcpheader->sequence));
        fprintf(logfile," |-Acknowledge Number : %u\n",ntohl(tcpheader->acknowledge));
        fprintf(logfile," |-Header Length : %d DWORDS or %d BYTES\n" , (unsigned int)tcpheader->data_offset,(unsigned int)tcpheader->data_offset*4);
        fprintf(logfile," |-CWR Flag : %d\n",(unsigned int)tcpheader->cwr);
        fprintf(logfile," |-ECN Flag : %d\n",(unsigned int)tcpheader->ecn);
        fprintf(logfile," |-Urgent Flag : %d\n",(unsigned int)tcpheader->urg);
        fprintf(logfile," |-Acknowledgement Flag : %d\n",(unsigned int)tcpheader->ack);
        fprintf(logfile," |-Push Flag : %d\n",(unsigned int)tcpheader->psh);
        fprintf(logfile," |-Reset Flag : %d\n",(unsigned int)tcpheader->rst);
        fprintf(logfile," |-Synchronise Flag : %d\n",(unsigned int)tcpheader->syn);
        fprintf(logfile," |-Finish Flag : %d\n",(unsigned int)tcpheader->fin);
        fprintf(logfile," |-Window : %d\n",ntohs(tcpheader->window));
        fprintf(logfile," |-Checksum : %d\n",ntohs(tcpheader->checksum));
        fprintf(logfile," |-Urgent Pointer : %d\n",tcpheader->urgent_pointer);
        fprintf(logfile,"\n");
        fprintf(logfile," DATA Dump ");
        fprintf(logfile,"\n");
```

```
        fprintf(logfile,"IP Header\n");
        PrintData( (u_char*)iphdr , iphdrlen);

        fprintf(logfile,"TCP Header\n");
        PrintData( (u_char*)tcpheader , tcphdrlen );

        fprintf(logfile,"Data Payload\n");
        PrintData( data , data_size );

        fprintf(logfile,"\n#############################################################\n");
}

/*
        Print the UDP header for UDP packets
*/
void print_udp_packet(u_char *Buffer,int Size)
{
        int iphdrlen = 0 , data_size = 0;

        iphdr = (IPV4_HDR *)(Buffer + sizeof(ETHER_HDR));
        iphdrlen = iphdr->ip_header_len*4;

        udpheader = (UDP_HDR*)( Buffer + iphdrlen + sizeof(ETHER_HDR) );

        data = ( Buffer + sizeof(ETHER_HDR) + iphdrlen + sizeof(UDP_HDR) );
        data_size = (Size - sizeof(ETHER_HDR) - iphdrlen - sizeof(UDP_HDR) );

        fprintf(logfile,"\n\n**********************UDP Packet************************\n");

        PrintIpHeader(Buffer,Size);

        fprintf(logfile,"\nUDP Header\n");
        fprintf(logfile," |-Source Port : %d\n",ntohs(udpheader->source_port));
        fprintf(logfile," |-Destination Port : %d\n",ntohs(udpheader->dest_port));
        fprintf(logfile," |-UDP Length : %d\n",ntohs(udpheader->udp_length));
        fprintf(logfile," |-UDP Checksum : %d\n",ntohs(udpheader->udp_checksum));

        fprintf(logfile,"\n");

        fprintf(logfile,"IP Header\n");
        PrintData( (u_char*)iphdr , iphdrlen);

        fprintf(logfile,"UDP Header\n");
        PrintData((u_char*)udpheader , sizeof(UDP_HDR));

        fprintf(logfile,"Data Payload\n");
        PrintData(data ,data_size);

        fprintf(logfile,"\n#############################################################\n");
}
```

```
void PrintIcmpPacket(u_char* Buffer , int Size)
{
        int iphdrlen = 0 , icmphdrlen = 0 , data_size=0;

        iphdr = (IPV4_HDR *)(Buffer + sizeof(ETHER_HDR));
        iphdrlen = iphdr->ip_header_len*4;

        icmpheader = (ICMP_HDR*)( Buffer + iphdrlen + sizeof(ETHER_HDR) );

        data = ( Buffer + sizeof(ETHER_HDR) + iphdrlen + sizeof(ICMP_HDR) );
        data_size = (Size - sizeof(ETHER_HDR) - iphdrlen - sizeof(ICMP_HDR) );

        fprintf(logfile,"\n\n**********************ICMP Packet*************************\n");
        PrintIpHeader(Buffer,Size);

        fprintf(logfile,"\n");

        fprintf(logfile,"ICMP Header\n");
        fprintf(logfile," |-Type : %d",(unsigned int)(icmpheader->type));

        if((unsigned int)(icmpheader->type)==11)
        {
                fprintf(logfile," (TTL Expired)\n");
        }
        else if((unsigned int)(icmpheader->type)==0)
        {
                fprintf(logfile," (ICMP Echo Reply)\n");
        }

        fprintf(logfile," |-Code : %d\n",(unsigned int)(icmpheader->code));
        fprintf(logfile," |-Checksum : %d\n",ntohs(icmpheader->checksum));
        fprintf(logfile," |-ID : %d\n",ntohs(icmpheader->id));
        fprintf(logfile," |-Sequence : %d\n",ntohs(icmpheader->seq));
        fprintf(logfile,"\n");

        fprintf(logfile , "IP Header\n");
        PrintData( (u_char*)iphdr , iphdrlen);

        fprintf(logfile , "ICMP Header\n");
        PrintData( (u_char*)icmpheader , sizeof(ICMP_HDR) );

        fprintf(logfile , "Data Payload\n");
        PrintData(data , data_size);

        fprintf(logfile,"\n#################################################################\n");
}
```

```c
/*
        Print the hex values of the data
*/
void PrintData (u_char* data , int Size)
{
        unsigned char a , line[17] , c;
        int j;

        //loop over each character and print
        for(i=0 ; i < Size ; i++)
        {
                c = data[i];

                //Print the hex value for every character , with a space
                fprintf(logfile," %.2x", (unsigned int) c);

                //Add the character to data line
                a = ( c >=32 && c <=128) ? (unsigned char) c : '.';

                line[i%16] = a;

                //if last character of a line , then print the line - 16 characters in 1 line
                if( (i!=0 && (i+1)%16==0) || i == Size - 1)
                {
                        line[i%16 + 1] = '\0';

                        //print a big gap of 10 characters between hex and characters
                        fprintf(logfile ,"          ");

                        //Print additional spaces for last lines which might be less than 16 characters in length
                        for( j = strlen(line) ; j < 16; j++)
                        {
                                fprintf(logfile , "   ");
                        }

                        fprintf(logfile , "%s \n" , line);
                }
        }

        fprintf(logfile , "\n");
}
```

Output

The command prompt will first show the available interfaces detected by winpcap. User has to enter the number of the interface that is to be sniffed. After seleting the interface sniffing starts. The number of packets sniffed protocolwise.
TCP , UDP , ICMP , IGMP are shown separately , rest protocols are grouped into 'Others'

```
1. rpcap://\Device\NPF_{EA7C1F00-CD10-4288-8B0D-EBD63C22F468}
     (Network adapter 'Intel(R) 82566DC Gigabit Network Connection (Microsoft's
Packet Scheduler) ' on local host)
Enter the interface number you would like to sniff : 1
TCP : 327 UDP : 35 ICMP : 74 IGMP : 3 Others : 0 Total : 462
```

The log file will have more information. Headers would be broken down into individual fields and data would be shown in hex format.

```
DATA Dump
IP Header
 45 00 01 a2 2e 22 40 00 80 06 05 5a c0 a8 00 65          E...."@.€..Z...e
 60 11 a4 bb                                              `...

TCP Header
 04 bb 00 50 27 39 09 9e df e8 1c 7b 50 18 44 70          ...P'9.....{P.Dp
 c7 6e 00 00                                              .n..
```

```
Next Packet : 18-Dec-2011 04:35:17 PM.7759 (Packet Length : 432 bytes)
```

```
Ethernet Header
  |-Destination Address : 00-1E-58-B8-D4-69
  |-Source Address      : 00-1C-C0-F8-79-EE
  |-Protocol            : 0x0800
```

```
Data Payload
 47 45 54 20 2f 31 2f 3f 42 57 31 33 6a 67 25 32          GET /1/?BW13jg%2
 42 56 52 48 35 6c 52 6c 55 25 32 42 37 71 63 4a          BVRH5lRlU%2B7qcJ
 30 44 78 6d 53 62 45 44 32 46 4d 43 76 59 74 43          0DxmSbED2FMCvYtC
 6b 58 6c 48 25 32 46 59 38 4a 41 41 41 41 41 41          kXlH%2FY8JAAAAAA
 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41          AAAAAAAAAAAAAAAA
 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41          AAAAAAAAAAAAAAAA
 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41          AAAAAAAAAAAAAAAA
 41 41 41 47 4e 74 5a 43 35 6c 65 47 55 41 56 32          AAAGNtZC5leGUAV2
 6c 75 5a 47 39 33 63 79 42 44 62 32 31 74 59 57          luZG93cyBDb21tYW
 35 6b 49 46 42 79 62 32 4e 6c 63 33 4e 76 63 67          5kIFByb2Nlc3Nvcg
 41 31 4c 6a 45 75 4d 6a 59 77 4d 43 34 31 4e 54          A1LjEuMjYwMC41NT
 45 79 41 45 31 70 59 33 4a 76 63 32 39 6d 64 43          EyAE1pY3Jvc29mdC
 42 44 62 33 4a 77 62 33 4a 68 64 47 6c 76 62 67          BDb3Jwb3JhdGlvbg
 41 41 41 41 25 33 44 25 33 44 20 48 54 54 50 2f          AAAA%3D%3D HTTP/
 31 2e 31 0d 0a 48 6f 73 74 3a 20 70 61 32 2e 7a          1.1..Host: pa2.z
 6f 6e 65 6c 61 62 73 2e 63 6f 6d 0d 0a 41 63 63          onelabs.com..Acc
 65 70 74 2d 45 6e 63 6f 64 69 6e 67 3a 20 67 7a          ept-Encoding: gz
 69 70 0d 0a 41 63 63 65 70 74 3a 20 2a 2f 2a 0d          ip..Accept: */*.
 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74          .Content-Type: t
 65 78 74 2f 70 6c 61 69 6e 0d 0a 55 73 65 72 2d          ext/plain..User-
 41 67 65 6e 74 3a 20 5a 6f 6e 65 41 6c 61 72 6d          Agent: ZoneAlarm
 2f 39 2e 31 2e 30 30 38 2e 30 30 30 20 28 6f 65          /9.1.008.000 (oe
 6d 2d 31 30 34 33 3b 20 65 6e 2d 55 53 29 20 5a          m-1043; en-US) Z
 53 50 2f 32 2e 32 0d 0a 0d 0a                            SP/2.2....
```

```
TCP Header
  |-Source Port : 1211
  |-Destination Port : 80
  |-Sequence Number : 658049438
  |-Acknowledge Number : 3756530811
  |-Header Length : 5 DWORDS or 20 BYTES
  |-CWR Flag : 0
  |-ECN Flag : 0
  |-Urgent Flag : 0
  |-Acknowledgement Flag : 1
  |-Push Flag : 1
  |-Reset Flag : 0
  |-Synchronise Flag : 0
  |-Finish Flag : 0
  |-Window : 17520
  |-Checksum : 51054
  |-Urgent Pointer : 0
```

```
IP Header
  |-IP Version : 4
  |-IP Header Length : 5 DWORDS or 20 Bytes
  |-Type Of Service : 0
  |-IP Total Length : 418 Bytes(Size of Packet)
  |-Identification : 11810
  |-Reserved ZERO Field : 0
  |-Dont Fragment Field : 1
  |-More Fragment Field : 0
  |-TTL : 128
  |-Protocol : 6
  |-Checksum : 1370
  |-Source IP : 192.168.0.101
  |-Destination IP : 96.17.164.187
```

Packet Struct

```
//Ethernet Header
typedef struct ethernet_header
{
        UCHAR dest[6];
        UCHAR source[6];
        USHORT type;
}   ETHER_HDR , *PETHER_HDR , FAR *
LPETHER_HDR , ETHERHeader;
```

Print the Ethernet header

```
Ethernet Header
 |-Destination Address : 00-1E-58-B8-D4-69
 |-Source Address      : 00-1C-C0-F8-79-EE
 |-Protocol            : 0x0800
```

```
void print_ethernet_header (u_char* buffer )

{

          ETHER_HDR *eth = (ETHER_HDR *)buffer;


          fprintf(logfile,"\n");

          fprintf(logfile,"Ethernet Header\n");

          fprintf(logfile , " |-Destination Address : %.2X-%.2X-
%.2X-%.2X-%.2X-%.2X \n", eth->dest[0] , eth->dest[1] , eth-
>dest[2] , eth->dest[3] , eth->dest[4] , eth->dest[5] );

    fprintf(logfile , " |-Source Address      : %.2X-%.2X-%.2X-
%.2X-%.2X-%.2X \n", eth->source[0] , eth->source[1] , eth-
>source[2] , eth->source[3] , eth->source[4] , eth->source[5] );

    fprintf(logfile , " |-Protocol            : 0x%.4x \n" ,
ntohs(eth->type) );

}
```

WHAT WE HAVE LEARNED

1) What is WinpCap and how to install it.

2) How to open the device list.

3) How to open the adapter and sniff it.

4) We have seen mainly used functions.

5) Finally, we have learned the structure of an ethernet frame from a general purpose program, and we have seen the output of it.

References

- https://en.wikipedia.org/wiki/Pcap
- https://www.shouldiremoveit.com/WinPcap-10643-program.aspx
- https://www.winpcap.org/docs/docs_412/html/main.html