Lecture 5: Packet Sniffing

1. Tool

tcpdump for Unix/Linux/macOS.

windump for windows

2. windump

From the iClass

-download WinPcap and install

-download WinDump

-run WinDump

3. usage (use tcpdump for macOS)

windump -AdDeflLnNOpqRStuUvxX -c count -C file\_size -F file -i interface

-s snaplen -T type -w file -W filecount expression

-A print each packet in ascii

-c exit after receiving count packets

-D print available network interfaces

-e print the link-level header on each dump line

-i Listen on "interface"

-n Don't convert addresses to names

-q short output format

-S print absolute, rather than relative, TCP sequence numbers

-s snarf "snaplen" bytes from each packet rather than the default of 68

-t Don't print a timestamp on each dump line

-w write the raw packets to "file" rather than parsing and printing them out. standard output is used if the file is "-"

-x print in hex

-xx same as -x but print link level header, too.

-X print in hex and ascii

-XX same as -X but print link level header, too

expression:

packets that make "expression" true will be captured

expression = [primitive+] constrained by and, or, not

primitive = [qualifier+] id

qualifier = type | dir | proto

type = host(default) | net | port | portrange

ex) host foo : packet for foo

net 128.3 : packet for 128.3.\*.\*

port 20 : packet for port 20

dir = src | dst | src or dst (default) | src and dst

ex) src foo : packet coming from foo

proto = ether | ip | arp | rarp | wlan | tcp | udp

if not specified all relevant protocols are captured

examples)

windump host sundown : print all in/out packets for sundown

windump host ace and not helios : print all packets between ace and any host except helios

windump -D : shows all available network interface

windump -eSXX -i 2 -s 80 port 9924 : show all in/out packets in interface 2 for port 9924 in hex including the link header. Capture max 80 bytes.

(in macOS: sudo tcpdump -eSXX -i 2 -s 80 port 9924)

4. Homework:

Follow the steps below to display the contents of the packets between the server and the client. Submit explanation of each field in those packets.

1) Make a client in your PC as follows.

(For macOS, use sftp to download cli.c from lab server and use it to connect to the server.)

a) Run Microsoft Visual Studio.

b) Create an empty project:

- Select file->new->projects->win32 console application

- Adjust “Location” for the project directory and give a project name (e.g. “proj1”)

and hit “Confirm” button

- Hit “Next” in the wizard window.

- Uncheck every box and check “Empty Project” box and hit “Finish” button.

c) Write a C++ source file

- Press the right mouse button on “proj1” symbol.

- Select Add->New Item->C++ file

- Give a file name (for example: main.cpp) and hit “Add” button.

- Copy and paste "wincli.cpp" code (given below). Adjust ip/port number.

d) Compile

- Select “project->proj1->manifest tools->input and output->include manifest”

and set “No”

- add ws2\_32.lib in project>properties>linker>input>additional dependencies>edit

- Select build->Solution Build

- You should see “Success 1” at the bottom of the compile window.

e) Run

- Run the server first.

- Run the client: select Debug->Execute without debugging

(\* if you have LINK error even with ws2\_32.lib included, add below at the top of your code:

#pragma comment (lib, "ws2\_32.lib")

\*)

2) Download win10pcap from iClass and install. Download windump.exe from iClass and run (in the command window; open command window as admin if needed) to monitor packets for specified port.

windump -D : check available network interfaces

windump -eSXX -i 2 -s 80 port 9924 : monitor packets at device 2 whose src or dest ports are 9924

(for wireshark: right mouse button click, run as admin)

(for MacOS, use tcpdump)

3) Run your server again. Run the client in your PC.

4) Find the first packet which is a SYN packet sent by the client to the server in the windump window. Extract all packet header information. Refer TCP packet structure in Section 6 below.

5) Analyze rest of the packets similarly.

6) Connect to www.inha.ac.kr and analyze SYN, S/ACK, ACK packets between the web browser and www.inha.ac.kr. You may need "-c num" option to capture the first num packets as below.

windump –eSXX –c 20 –i 2 –s 80 host www.inha.ac.kr

7) Click login menu(로그인) and enter id and password. Find the packet that contains your login ID and password. To capture login ID and password, make the capture size larger, e.g. 3000. Use –w option to save the result in a file (e.g. pktout) and use –r option to read packets from a file.

windump –eSXX –w pktout –i 2 –s 3000 host www.inha.ac.kr

windump –eSXX –r pktout > x

vi x

8) Connect to portal.inha.ac.kr and do the same thing as in Problem 6) and 7).

5. wincli.cpp

#include "winsock2.h"

#include "ws2tcpip.h"

#include "stdio.h"

#define SERVER\_PORT 9924 // server port number

#define BUF\_SIZE 4096 // block transfer size

#define QUEUE\_SIZE 10

#define IPAddress "165.246.38.152" // server IP address

int main()

{

WORD wVersionRequested;

WSADATA wsaData;

SOCKADDR\_IN target; //Socket address information

SOCKET s;

int err;

int bytesSent;

char buf[100];

//--- INITIALIZATION -----------------------------------

wVersionRequested = MAKEWORD( 1, 1 );

err = WSAStartup( wVersionRequested, &wsaData );

if ( err != 0 ) {

printf("WSAStartup error %ld", WSAGetLastError() );

WSACleanup();

return false;

}

//------------------------------------------------------

//---- Build address structure to bind to socket.--------

target.sin\_family = AF\_INET; // address family Internet

target.sin\_port = htons (SERVER\_PORT); //Port to connect on

inet\_pton(AF\_INET, IPAddress, &(target.sin\_addr.s\_addr)); // target IP

//--------------------------------------------------------

// ---- create SOCKET--------------------------------------

s = socket (AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); //Create socket

if (s == INVALID\_SOCKET)

{

printf("socket error %ld" , WSAGetLastError() );

WSACleanup();

return false; //Couldn't create the socket

}

//---------------------------------------------------------

//---- try CONNECT -----------------------------------------

if (connect(s, (SOCKADDR \*)&target, sizeof(target)) == SOCKET\_ERROR)

{

printf("connect error %ld", WSAGetLastError() );

WSACleanup();

return false; //Couldn't connect

}

//-------------------------------------------------------

//---- SEND bytes -------------------------------------------

printf("enter a string to send to server\n");

gets\_s(buf, 99);

bytesSent = send( s, buf, strlen(buf), 0 ); // use "send" in windows

printf( "Bytes Sent: %ld \n", bytesSent );

// now receive

int n;

n=recv(s, buf, 50, 0); // read max 50 bytes

buf[n]=0; // make a string

printf("received: %s\n", buf);

//--------------------------------------------------------

closesocket( s );

WSACleanup();

return 0;

}

6. TCP packet structure

packet =

Ethernet header (14)+ IP header(20) + TCP header(20 or greater) + data + Ethernet trailer(4)

Ethernet header=Dest addr(6) + Src addr(6) + type(2)

Dest addr: hardware address of the destination network interface

Src addr: hardware address of the source network interface

type : 0x0800 (IP packet), 0x0806(ARP packet), 0x8035(RARP packet), .....

IP header=

version(4 bit)+header length(4 bit)+type of service(8 bit)+total length(16 bit)

+identification(16 bit)+flags(3 bit)+fragment offset(13 bit)

+time to live(8 bit)+protocol(8 bit)+header checksum(16 bit)

+source IP address(32 bit)

+dest IP address(32 bit)

header length = number of 32-bit words in the header. normally 5 (5\*4 = 20 bytes)

TOS(Type Of Service)=unused bits(3)+TOS(4)+0

TOS=minimize delay(1)+maximize throughput(1)+

maximize reliability(1)+minimize monetary cost(1)

total length=total bytes in the packet (IP header + TCP header + data + trailer).

16-bit field ==> max IP packet size is 2\*\*16-1 = 65535

identification :unique number for each IP packet. When an IP packet is fragmented

all fragments have the same identification number to indicate they belong

to the same IP packet

fragment offset : show the position of this fragment in the original IP packet before

fragmentation

flags=0+DF(1)+MF(1)

DF: don't fragment this fragment further

MF: more fragment after this meaning this is not the final fragment

time to live(TTL): max routers this packet can pass. if ttl=8, after 8 routers, this packet will

be dropped.

protocol: tcp, udp, icmp, igmp

source/dest IP address: 32 bit. dotted expression shows each byte in decimal seperated

by '.'

TCP header =

source port number(16)+dest port number(16)

+ sequence number(32)

+ acknowledgment number(32)

+ header length(4)+reserved(6)+URG+ACK+PSH+RST+SYN+FIN+window size(16)

+ TCP checksum(16)+urgent pointer(16)

sequence number=ISN(Initial Sequence Number)

+ (# of bytes sent so far in this tcp connection)

acknowledgment number=next sequence number expected from the other part

7. TCP protocol

server: waits in accept()

client: calls connect()

client -> server: SYN packet

server-> client: SYN/ACK packet

client-> server:ACK packet

Now connection is established, and the client and server can send/receive data.