lecture 7: Sending a SYN packet with winpcap

1. algorithm

..............

fp = pcap\_open(..........);

unsigned char packet[65535]; // max packet size is 65535

build\_packet(packet);

pcap\_sendpacket(fp, packet, ......);

2. build\_packet() : ethernet header, ip header, tcp header

1) Build ethernet header

Define the structure for ethernet header.

struct ether\_addr {

unsigned char ether\_addr\_octet[6];

};

struct ether\_header {

struct ether\_addr ether\_dhost;

struct ether\_addr ether\_shost;

unsigned short ether\_type; // 0x0800 for IP

};

struct ether\_header \*myeh; // create a etherhead structure

Now fill in myeh. First make myeh point to the start of the packet buffer.

myeh = (struct ether\_header \*)packet;

And fill in the destination mac address.

myeh->ether\_dhost.ether\_addr\_octet[0]=dest\_mac[0];

myeh->ether\_dhost.ether\_addr\_octet[1]=dest\_mac[1];

..............

myeh->ether\_dhost.ether\_addr\_octet[5]=dest\_mac[5];

And the source mac address.

myeh->ether\_shost.ether\_addr\_octet[0]=src\_mac[0];

myeh->ether\_shost.ether\_addr\_octet[1]=src\_mac[1];

..............

myeh->ether\_shost.ether\_addr\_octet[5]=src\_mac[5];

MAC address of a network device is always 6 bytes. src\_mac is the MAC for the PC. dest\_mac is the MAC of the gateway computer. Get these addresses from captured packets.

Finally fill in the ethernet header type.

myeh->ether\_type = htons(0x0800); // must be in network byte order

2) Build ip header

Define the structure for the ip header.

struct ip\_hdr{

unsigned char ip\_header\_len:4;

unsigned char ip\_version:4;

unsigned char ip\_tos;

unsigned short ip\_total\_length;

unsigned short ip\_id;

unsigned char ip\_frag\_offset:5;

unsigned char ip\_more\_fragment:1;

unsigned char ip\_dont\_fragment:1;

unsigned char ip\_reserved\_zero:1;

unsigned char ip\_frag\_offset1;

unsigned char ip\_ttl;

unsigned char ip\_protocol;

unsigned short ip\_checksum;

unsigned int ip\_srcaddr;

unsigned int ip\_destaddr;

};

Make a variable for the ip header.

struct ip\_hdr \*myih;

Make it point to the proper location in packet buffer.

myih = (struct ip\_hdr \*)(packet + 14);

Now fill in myih.

myih->ip\_header\_len=5;

myih->ip\_version = 4;

.............

myih->ip\_checksum = 0;

myih->ip\_checksum = in\_checksum((unsigned short \*)myih, 20);

in\_checksum computes the checksum for the given bytes as follows:

unsigned short in\_checksum(unsigned short \*ptr,int nbytes) {

register long sum;

unsigned short oddbyte;

register short answer;

sum=0;

while(nbytes>1) {

sum+=\*ptr++;

nbytes-=2;

}

if(nbytes==1) {

oddbyte=0;

\*((u\_char\*)&oddbyte)=\*(u\_char\*)ptr;

sum+=oddbyte;

}

sum = (sum>>16)+(sum & 0xffff);

sum = sum + (sum>>16);

answer=(SHORT)~sum; // use “short” in MacOS

return(answer);

}

3) Build tcp header

Define a structure for the tcp header:

struct tcp\_hdr{

unsigned short source\_port;

unsigned short dest\_port;

unsigned int sequence;

unsigned int acknowledge;

unsigned char ns:1;

unsigned char reserved\_part1:3;

unsigned char data\_offset:4;

unsigned char fin:1;

unsigned char syn:1;

unsigned char rst:1;

unsigned char psh:1;

unsigned char ack:1;

unsigned char urg:1;

unsigned char ecn:1;

unsigned char cwr:1;

unsigned short window;

unsigned short checksum;

unsigned short urgent\_pointer;

};

Define a variable to point to this structure.

struct tcp\_hdr \*myth;

Make it point to the proper location in the packet buffer.

myth = (struct tcp\_hdr \*)(packet + 14 + 20);

Now fill in myth.

myth->source\_port=htons(src\_port);

myth->dest\_port=htons(dest\_port);

...............

myth->window=htons(0x4000);

myth->checksum=0;

myth->urgent\_pointer=0;

Checksum has to be computed properly. The ckecksum algorithm is applied to "pseudo header + tcp header + data".

Computing psudo header

struct pseudo\_header{

unsigned int source\_address;

unsigned int dest\_address;

unsigned char placeholder;

unsigned char protocol;

unsigned short tcp\_length;

};

struct pseudo\_header psh;

// to use inet\_pton(), include "winsock2.h" and "ws2tcpip.h" in windows

// in MacOS, include <arpa/inet.h>

inet\_pton(AF\_INET, "211.110.41.36", &(psh.source\_address)); // ip of your pc

inet\_pton(AF\_INET, "165.246.38.151", &(psh.dest\_address)); // dest ip

psh.placeholder=0; // reserved

psh.protocol=6; // protocol number for tcp

psh.tcp\_length=htons(tcp\_header\_len); // store multi byte number in network byte order

Now build pseudo header + tcp header.

unsigned char \*seudo;

seudo = (unsigned char \*)malloc(sizeof(struct pseudo\_header)+tcp\_header\_len);

memcpy(seudo, &psh, sizeof(struct pseudo\_header));

memcpy(seudo+sizeof(struct pseudo\_header), myth, tcp\_header\_len);

Compute the checksum.

myth->checksum=

in\_checksum((unsigned short \*)seudo, sizeof(struct pseudo\_header)+tcp\_header\_len);

3. Send the packet

if (pcap\_sendpacket(fp, packet, 14+20+tcp\_header\_len) !=0){

printf("err in packet send:%s\n",pcap\_geterr(fp));

}

4. Homework

1) Send a SYN packet to the server.

1-1)

- Modify your sniffer such that it can send a SYN packet instead of sniffing.

-- break out of the while loop after capturing the first SYN packet.

-- display the packet in raw bytes.

-- kill the server and the client (manually)

-- run the original sniffer

-- rerun the server

-- send the captured SYN packet to the server

-- check if you can see this packet in the original sniffer

-- check if you can see the ACK packet from the server.

\*If the captured SYN packet shows checksum=0, do not send as it is: you have to compute the checksum by yourself (checksum offload case).

1-2)

- Modify the sniffer further such that it re-computes ip and tcp checksum.

-- break out of the while loop after capturing the first SYN packet.

-- copy them into another buffer: pkt\_data=>packet

-- set ip\_check\_sum and tcp\_check\_sum to zero

-- recompute ip\_check\_sum

-- recompute tcp\_check\_sum

-- display the packet in raw bytes. this should be same as pkt\_data

-- kill the server and the client (manually)

-- run the original sniffer

-- rerun the server

-- send packet to the server

-- check if you can see this packet in the original sniffer

-- check if you can see the ACK packet from the server.

2) Implement a stealth scanner.

- Modify the packet sender such that it sends a SYN packet to all possible ports in the server. Detect which ports are live by finding out those who respond with SYN/ACK. You may need seperate sniffer for such detection.

(\* If you have "fatal bad memory block" error, change win10pcap to wpcap. Delete win10pcap and reinstall wpcap.)