**네트워크 개발 - PCAP**

BoB 8기 보안제품개발 정동욱

< 목차 >

1) 문제 정의

2) 문제 조건

3) 코드 작성 및 분석

4) 결과 및 느낀점

1) 문제 정의

[리포트] pcap을 이용하여 송수신되는 packet의 다음 값을 출력하는 프로그램을 작성하라. eth.smac, eth.dmac / ip.sip, ip.dip / tcp.sport, tcp.dport / data(최대 10바이트까지)

syntax : pcap\_test <interface>

example : pcap\_test wlan0

2) 문제 조건

1] gitlab 사이트에 있는 pcap\_test.zip 파일을 skeleton code로 하여 printf하는 부분만 수정해서 제출하는 것 추천.

2] ETH 다음에 IP가 오는지 확인하는 코드가 들어갈 것(정확한 작동 방식은 구글링해서 익힐 것).

3] IP 다음에 TCP가 오는지 확인하는 코드가 들어갈 것(정확한 작동 방식은 구글링해서 익힐 것).

4] TCP Data의 위치와 크기를 알아내는 코드가 들어갈 것(정확한 작동 방식은 구글링해서 익힐 것).

5] TCP Data는 최대 10바이트까지만 찍을 것(최대라고 했지 무조건 10은 아님. 10보다 데이터의 크기가 작을 수도 있음).

6] mac, ip, port를 출력할 때 코드의 중복이 없도록 별도의 함수를 만들어 사용할 것(일요일 실습 내용 참고).

7] git repository에는 임시 파일(*.o, 실행 파일)은 올리지 말 것. 프로젝트 파일(*.pro)은 올릴 것.

8] 소스 코드는 C/C++로 작성할 것. scapy나 libtins와 같이 wrapping이 잘 되어 있는 라이브러리는 사용하지 말 것.

3) 코드 작성 및 분석

#include <pcap.h>

#include <stdio.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdint.h>

#include <string.h>

#include <stdlib.h>

char WIP[2]; // what\_is\_protocol

struct eth\_addr

{

u\_char eth\_address\_octets[6];

// A ethernet address has 6 octets.

};

struct eth\_header // Size : 14 Bytes

{

struct eth\_addr eth\_destination\_host; // 6 Octet

struct eth\_addr eth\_source\_host; // 6 Octet

u\_short eth\_type; // 2 Octet

};

struct ip\_header // by IP header order & Size : minimum 20 Bytes

{

u\_char version:4; // 0.5 Octet

u\_char header\_len:4; // 0.5 Octet

u\_char type\_of\_service; // 1 Octet

u\_short total\_length; // 2 Octet

u\_short identification; // 2 Octet

u\_char flags; // 0.5 Octet

u\_char frag\_offset:5; // 0.5 Octet

u\_char more\_fragment:1; // 0.5 Octet

u\_char dont\_fragment:1; // 0.5 Octet

u\_char reserved\_zero:1; // 0.5 Octet

u\_char time\_to\_live; // 1 Octet

u\_char protocol; // 1 Octet

u\_short header\_checksum; // 2 Octet

struct in\_addr source\_address; // 4 Octet

struct in\_addr destination\_address; // 4 Octet

};

struct tcp\_header // by TCP header order & Size : minimum 20 Bytes

{

u\_short source\_port; // 2 Octet

u\_short dest\_port; // 2 Octet

u\_int sequence; // 4 Octet

u\_int acknowledge; // 4 Octet

u\_char header\_length:1; // 1 Octet

u\_char reserved\_part:3; // 1 Octet

u\_char data\_offset:4; // 0.5 Octet

u\_char fin:1;

u\_char syn:1;

u\_char rst:1;

u\_char psh:1;

u\_char ack:1;

u\_char urg:1;

u\_char ece:1;

u\_char cwr:1; // total 1.5 Octet

u\_short window; // 2 Octet

u\_short checksum; // 2 Octet

u\_short urgent\_pointer; // 2 Octet

};

struct udp\_header

{

u\_short source\_port; // 2 Octet

u\_short dest\_port; // 2 Octet

u\_short length; // 2 Octet

u\_short checksum; // 2 Octet

};

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

uint8\_t Print\_Eth(const u\_char\* Packet\_DATA){

struct eth\_header\* EH = (struct eth\_header\*)(Packet\_DATA);

uint8\_t EH\_length = (uint8\_t)(sizeof(EH));

u\_short ethernet\_type;

ethernet\_type = ntohs(EH->eth\_type);

if(ethernet\_type != 0x0800){

printf("Ethernet type is not IP\n");

return 0;

} // IP CHECK

printf("[Source] <MAC> Address : %02x:%02x:%02x:%02x:%02x:%02x:\n",

EH->eth\_source\_host.eth\_address\_octets[0],

EH->eth\_source\_host.eth\_address\_octets[1],

EH->eth\_source\_host.eth\_address\_octets[2],

EH->eth\_source\_host.eth\_address\_octets[3],

EH->eth\_source\_host.eth\_address\_octets[4],

EH->eth\_source\_host.eth\_address\_octets[5]);

printf("[Destination] <MAC> Address : %02x:%02x:%02x:%02x:%02x:%02x:\n",

EH->eth\_destination\_host.eth\_address\_octets[0],

EH->eth\_destination\_host.eth\_address\_octets[1],

EH->eth\_destination\_host.eth\_address\_octets[2],

EH->eth\_destination\_host.eth\_address\_octets[3],

EH->eth\_destination\_host.eth\_address\_octets[4],

EH->eth\_destination\_host.eth\_address\_octets[5]);

return EH\_length;

}

char\* Print\_IP(const u\_char\* Packet\_DATA){

struct ip\_header\* IH = (struct ip\_header\*)(Packet\_DATA);

// IP Check

if(IH->header\_len == 0) return 0;

if(IH->version < 4 && IH->version > 9) return 0;

// 4 : IP

// 5 : ST

// 6 : SIP, SIPP, IPv6

// 7 : TP/IX

// 8 : PIP

// 9 : TUBA

//printf("TTL : %x\n", IH->time\_to\_live);

//printf("protocol : %x\n", IH->protocol);

sprintf(WIP, "%x", IH->protocol);

//printf("WIP : %s\n", WIP);

printf("[Source] <IP> Address : %s\n", inet\_ntoa(IH->source\_address));

printf("[Destination] <IP> Address : %s\n", inet\_ntoa(IH->destination\_address));

return WIP;

}

int print\_TCP(const u\_char\* Packet\_DATA){

struct tcp\_header\* TH = (struct tcp\_header\*)(Packet\_DATA);

// TCP check

if(TH->data\_offset < 4) return 0;

char\* sp = (char\*)malloc(sizeof(TH->source\_port));

sprintf(sp, "%d", ntohs(TH->source\_port));

char\* dp = (char\*)malloc(sizeof(TH->dest\_port));

sprintf(dp, "%d", ntohs(TH->dest\_port));

//printf("sp : %s\n", sp);

//printf("dp : %s\n", dp);

if((!strcmp(sp, "443")) || (!strcmp(dp, "443"))){

printf("TCP SSL(HTTPS) protocol\n");

}

else if((!strcmp(sp, "25")) || (!strcmp(dp, "25"))){

printf("TCP SMTP protocol\n");

}

else if((!strcmp(sp, "53")) || (!strcmp(dp, "53"))){

printf("TCP DNS protocol\n");

}

else if((!strcmp(sp, "80")) || (!strcmp(dp, "80"))){

printf("TCP HTTP protocol\n");

}

else if((!strcmp(sp, "22")) || (!strcmp(dp, "22"))){

printf("TCP SSH protocol\n");

}

else if((!strcmp(sp, "23")) || (!strcmp(dp, "23"))){

printf("TCP Telnet protocol\n");

}

else if((!strcmp(sp, "111")) || (!strcmp(dp, "111"))){

printf("TCP RPC protocol\n");

}

printf("[Source] <Port> Number : %d\n", ntohs(TH->source\_port));

printf("[Destination] <Port> Number : %d\n", ntohs(TH->dest\_port));

return ((TH->data\_offset) \* 4);

}

int print\_UDP(const u\_char\* Packet\_DATA){

struct udp\_header\* UH = (struct udp\_header\*)(Packet\_DATA);

char\* sp = (char\*)malloc(sizeof(UH->source\_port));

sprintf(sp, "%d", ntohs(UH->source\_port));

char\* dp = (char\*)malloc(sizeof(UH->dest\_port));

sprintf(dp, "%d", ntohs(UH->dest\_port));

//printf("sp : %s\n", sp);

//printf("dp : %s\n", dp);

if((!strcmp(sp, "80")) || (!strcmp(dp, "80"))){

printf("UDP HTTP protocol\n");

}

else if((!strcmp(sp, "161")) || (!strcmp(dp, "161"))){

printf("UDP SNMP protocol\n");

}

else if((!strcmp(sp, "111")) || (!strcmp(dp, "111"))){

printf("UDP RPC protocol\n");

}

printf("[Source] <Port> Number : %d\n", ntohs(UH->source\_port));

printf("[Destination] <Port> Number : %d\n", ntohs(UH->dest\_port));

return (UH->length);

}

void print\_Data(const u\_char\* Packet\_DATA){

for(int i = 0; i < 10; i++) printf("%c", Packet\_DATA[i]);

printf("\n");

}

void usage() {

printf("syntax: pcap\_test <interface>\n");

printf("sample: pcap\_test wlan0\n");

}

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

if (argc != 2) {

usage();

return -1;

}

char\* dev = argv[1];

char errbuf[PCAP\_ERRBUF\_SIZE];

pcap\_t\* handle = pcap\_open\_live(dev, BUFSIZ, 1, 1000, errbuf);

if (handle == NULL) {

fprintf(stderr, "couldn't open device %s: %s\n", dev, errbuf);

return -1;

}

while (true) {

struct pcap\_pkthdr\* header;

const u\_char\* packet;

int res = pcap\_next\_ex(handle, &header, &packet);

if (res == 0) continue;

if (res == -1 || res == -2) break;

printf("########################\n\n");

printf("-- %u Bytes captured --\n\n", header->caplen);

printf("-------\_Ethernet\_-------\n");

uint8\_t tmp = 0; // Ethernet header size

tmp = Print\_Eth(packet);

//printf("packet : %d\n", tmp);

if(tmp > 14) break;

printf("\n");

printf("----------\_IP\_----------\n");

packet += 14;

char\* tmp2; // IP protocol type

int WIP = 0; // protocol's header size

tmp2 = Print\_IP(packet);

printf("\n");

//printf("tmp2 = %s\n", tmp2);

if(!strcmp(tmp2, "6")){ // TCP header : 20 Bytes

WIP = 20;

}

else if(!strcmp(tmp2, "11") || !strcmp(tmp2, "1")){

WIP = 8; // UDP header & ICMP header : 8 Bytes

}

else if(!strcmp(tmp2, "84")){

WIP = 4; // SCTP header : 4Bytes

}

if(!strcmp(tmp2, "6"))

printf("---------\_TCP\_---------\n");

else if(!strcmp(tmp2, "11"))

printf("---------\_UDP\_---------\n");

else

printf("--------\_Protocol\_--------\n");

packet += 20;

if(!strcmp(tmp2, "6"))

print\_TCP(packet);

else if(!strcmp(tmp2, "11"))

print\_UDP(packet);

else

printf("No Header Data here for this protocol!\n");

//printf("WIP : %d\n", WIP);

printf("\n");

printf("---------\_DATA\_---------\n");

packet += WIP;

if((!strcmp(tmp2, "6")) || (!strcmp(tmp2, "11")))

print\_Data(packet);

else

printf("No Protocol Data here!\n");

printf("\n");

printf("########################\n");

printf("\n\n");

}

pcap\_close(handle);

return 0;

}

4) 결과 사진

1] TCP Test

- tcpreplay tcp



- TCP packet 1



- TCP packet 2



- TCP packet 3



- TCP packet 4



- TCP packet 5



- TCP packet 6



2) UDP test

- tcpreplay udp



- UDP packet 1



5) 결론 및 느낀점

- wireshark로 패킷 캡쳐 했을 때 원리를 이용해서 C언어로 패킷 켭쳐하니까 더 구체적으로 배우게 되었습니다. 각 레이어의 프로토콜 헤더들의 구성 요소들이 어떤 방식으로 동작하고 의미가 무엇인지 알게 되었습니다. 네트워크 공부에 많은 흥미를 느끼게 해준 과제였습니다.

- 감사합니다