HW2

2013312343 이상헌

**1. Wireshark에서 packet을 파일로 추출하기**

- Wireshark에서 Ethernet으로 Packet을 분석한 뒤에, 분석 결과를 .pcap 확장자 파일로 저장하였다.

**2. 소스 코드**

- 먼저, 저장한 .pcap 파일을 불러오고, 패킷의 파일 헤더, 패킷 헤더 및 패킷을 순서대로 읽고 구조상대로 순차적으로 저장하였다.

- 이후 형식에 맞도록 출력하였다. 주의할 점은 엔디안 방식에 따른 바이트 오더의 다름이다. Windows10에서 분석 및 구현하였고 인텔 프로세스에서는 주로 리틀 엔디안 바이트 오더를 사용한다. 또한 네트워크 상에서 표준으로 이용되는 프로토콜은 네트워크 바이트 오더인 빅 엔디안으로 생각하였다. 이를 기준으로 바이트 오더를 바꾸는 함수를 따로 구현하고(ntohs\_) 이를 통해 수치의 바이트 오더를 고려하여 출력하였다.

- 아래는 소스 코드 전문이다.

#include <stdio.h>

#include <time.h>

#include <WinSock2.h>

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

#define MAGIC 0xa1b2c3d4

#define MAX\_P 30 // packet의 수가 많으면 결과를 출력하는 데 많은 시간이 소요되기 때문에, 한계를 설정하였다.

#define MAC\_ADDR\_LEN 6

#define DF(frag) (frag & 0x40)

#define MF(frag) (frag & 0x20)

#define FRAG\_OFFSET(frag) (ntohs(frag) & (~0x6000))

typedef struct p\_file\_header\_{

int magic;

unsigned short major\_ver;

unsigned short minor\_ver;

int thiszone;

unsigned sigfigs;

unsigned snaplen;

unsigned linktype;

}p\_file\_header; // 파일 헤더의 형식이다. 따로 출력할 필요는 없지만, 파일에서 이 부분 또한 읽어야 하기 때문에 구조체를 구현하고 저장만 하였다.

typedef struct timeval\_ {

long timesec;

long timeusec;

}timeval; // 패킷 헤더의 시간은 두 가지로 저장된다. 초 단위와 마이크로 초단위이다.

typedef struct p\_header\_ {

timeval time;

unsigned int caplen;

unsigned int actlen;

}p\_header; // 패킷 헤더의 내용이다. caplen은 분석 중에 캡쳐한 길이를 의미하고, actlen은 실제 패킷의 길이를 의미한다.

typedef struct ethernet\_ {

unsigned char dest\_mac[MAC\_ADDR\_LEN];

unsigned char src\_mac[MAC\_ADDR\_LEN];

unsigned short type;

}ethernet; // Ethernet 프로토콜 스택 구조에 맞게 구조체를 정의한다. Ethernet 프로토콜 스택 헤더는 목적지 MAC주소와 발신지 MAC 주소, 그리고 프로토콜 타입으로 구성된다.

typedef struct ip\_header\_{

unsigned char hlen : 4;

unsigned char version : 4;

unsigned char service;

unsigned short tot\_len;

unsigned short id;

unsigned short frag;

unsigned char time\_to\_live;

unsigned char protocol;

unsigned short checksum;

unsigned int src\_addr;

unsigned int dest\_addr;

}ip\_header; // ip 헤더의 구조체 구성이다. hlen과 version은 호스트 바이트 오더가 little endian이라는 가정을 하고 순서를 바꾸었다.

int Parsing(FILE \*fp);

void PacketParse(FILE \*fp);

void ViewPHeader(p\_header \*ph);

void ViewMac(unsigned char \*mac);

unsigned short ntohs\_(unsigned short value);

void ViewEthernet(char \*buffer);

void ViewIP(char \*buffer);

p\_header pheaders[MAX\_P];

int pcount = 0;

int Parsing(FILE \*fp)

{

p\_file\_header pfh;

fread(&pfh, sizeof(pfh), 1, fp);

PacketParse(fp);

return 0;

} // 먼저 오픈한 파일의 내용을 읽으면서 p\_file\_header 구조체에 파일 헤더의 구성요소의 값을 순서대로 저장한다.

void PacketParse(FILE \*fp) {

char buffer[65536];

p\_header \*ph = pheaders;

while (feof(fp) == 0) {

if (fread(ph, sizeof(p\_header), 1, fp) != 1) {

break;

}

if (pcount == MAX\_P) break;

ViewPHeader(ph);

fread(buffer, 1, ph->caplen, fp);

ViewEthernet(buffer);

ph++;

printf("\n");

}

} // 이후에 각 패킷의 헤더를 읽고, 값을 저장하는 방식을 반복하였다. buffer에 저장한 후에 이를 미리 구현한 구조체에 순차적으로 저장하고, 값을 출력한다.

void ViewPHeader(p\_header \*ph) {

pcount++;

printf("\nFrame %d: time: %08d:%06d , capture length: %u bytes , actual length: %u bytes\n", pcount, ph->time.timesec, ph->time.timeusec, ph->caplen, ph->actlen);

} // 패킷 헤더의 구성 요소 및 내용을 출력한다.

void ViewMac(unsigned char \*mac) {

int i;

for (i = 0; i < MAC\_ADDR\_LEN-1; i++) {

printf("%02x:", mac[i]);

}

printf("%02x", mac[i]);

} // MAC 주소를 형식에 맞게 출력하는 함수.

unsigned short ntohs\_(unsigned short value) {

return(value << 8) | (value >> 8);

} // endian을 고려하여 값을 변경해주는 함수이다.

void ViewEthernet(char \*buffer) {

ethernet \*ph = (ethernet\*)buffer;

printf("Src MAC address: "); ViewMac(ph->src\_mac); printf(" -> ");

printf("Dst MAC address: "); ViewMac(ph->dest\_mac);

if (ntohs\_(ph->type) == 0x800)

ViewIP(buffer + sizeof(ethernet));

} // Ethernet 프로토콜의 내용을 구조체에 순차적으로 저장하고, 송, 수신 MAC 주소를 출력한다.

void ViewIP(char \*buffer) {

IN\_ADDR addr;

ip\_header \*ip = (ip\_header \*)buffer;

addr.s\_addr = ip->src\_addr;

printf("\nSrc IP address: %s -> ",inet\_ntoa(addr));

addr.s\_addr = ip->dest\_addr;

printf("Dst IP address: %s\n", inet\_ntoa(addr));

switch (ip->protocol) {

case 1: printf("Protocol is ICMP. "); break;

case 2: printf("Protocol is IGMP. "); break;

case 6: printf("Protocol is TCP. "); break;

case 17: printf("Protocol is UDP. "); break;

case 89: printf("Protocol is OSPF. "); break;

default: printf("No support protocol. "); break;

}

printf("IP header length: %d bytes.\n", ip->hlen \* 4);

printf("id in decimal: %d, ", ntohs\_(ip->id));

printf("Frag-offset: %d, ", FRAG\_OFFSET(ip->frag));

if (DF(ip->frag)) printf("Flag: Don't Fragment.\n");

else {

if (MF(ip->frag) == 0) printf("Flag: Last Fragment.\n");

}

} // IPv4 프로토콜을 분석하고, 저장한 후에 이에 대한 내용을 출력하는 함수이다.

int main() {

char fname[300];

FILE \*fp = 0;

printf("파일 명을 입력하시오 : ");

gets\_s(fname, sizeof(fname));

fopen\_s(&fp, fname, "rb");

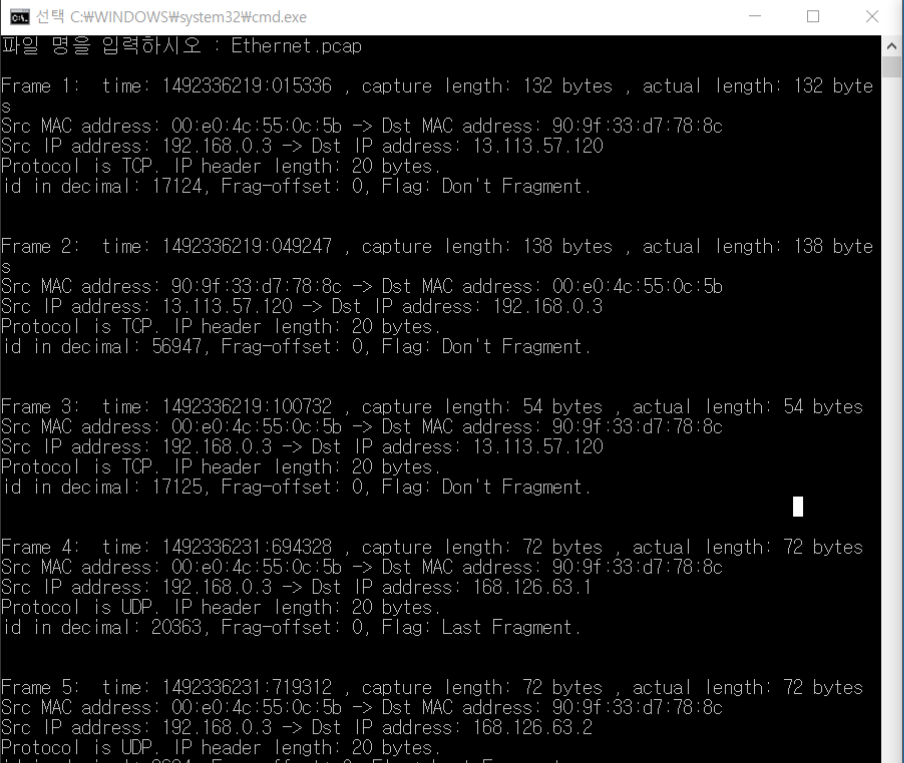
Parsing(fp);

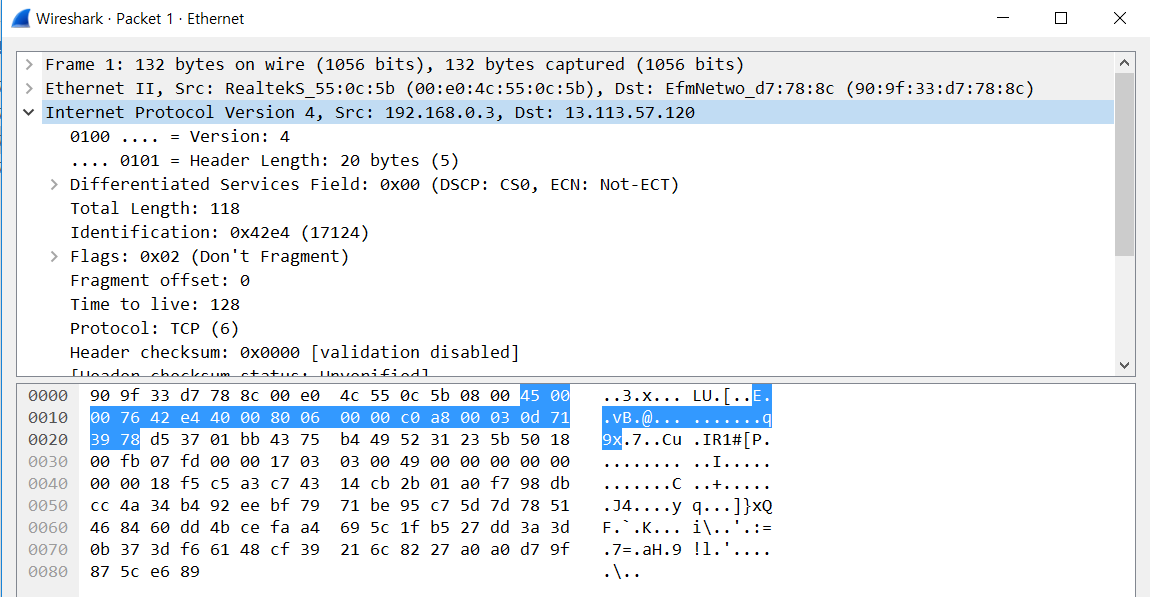
fclose(fp);

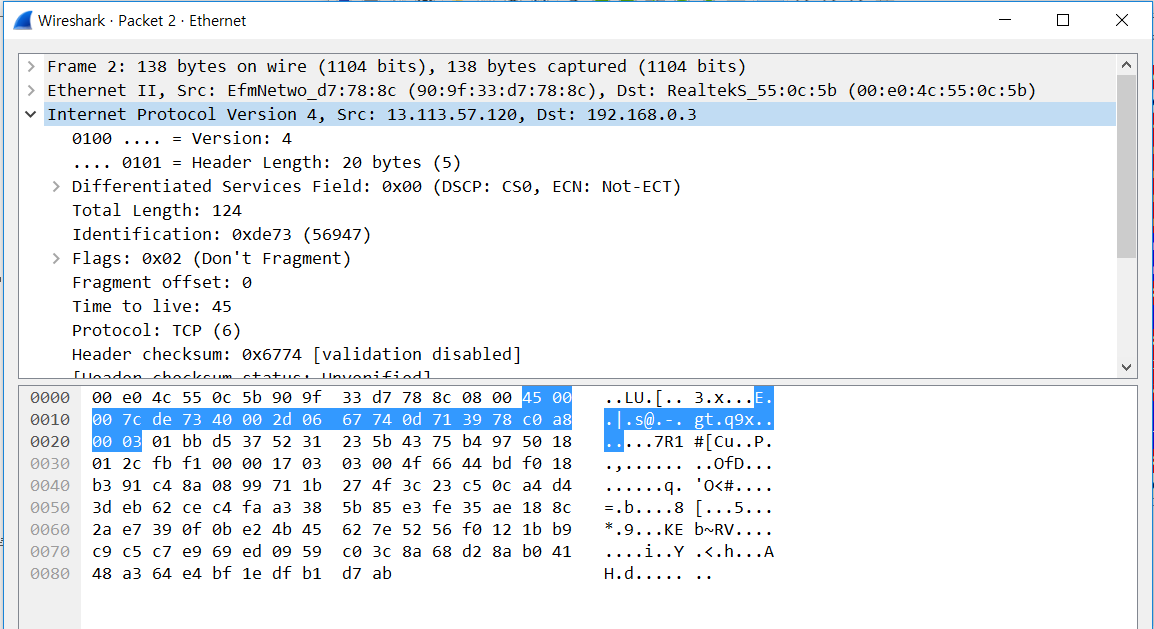
return 0;

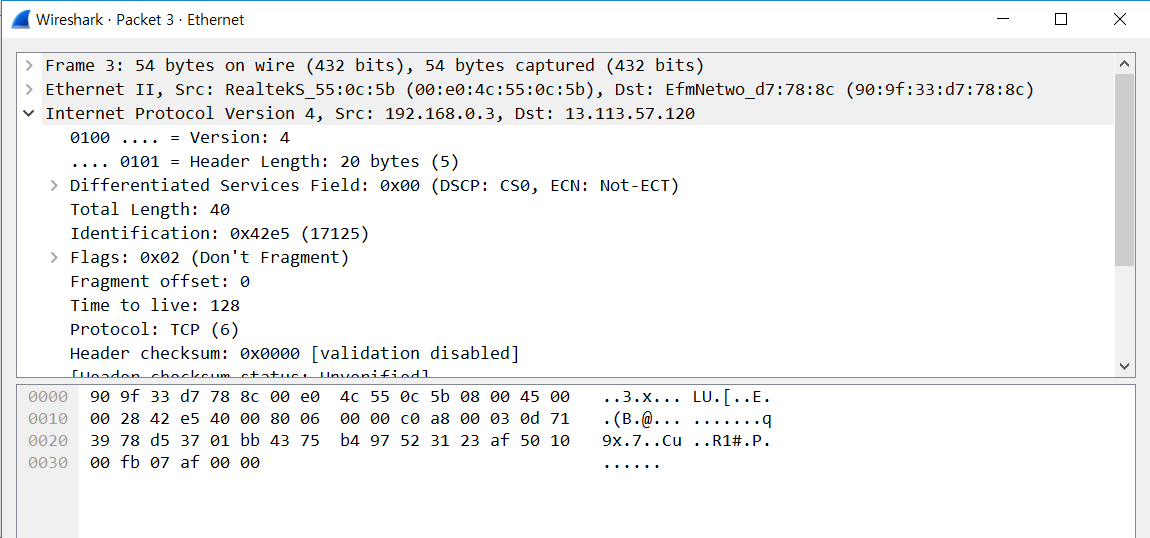
} // main 함수로, 파일을 열고 읽은 후에 Parsing 함수를 이용하여 출력한다.

**2. 실행 결과**









- 위의 그림은 소스 코드 실행 결과 및 Wireshark의 분석 결과를 비교한 것이다. 각 Frame마다 항목들이 소스 코드를 실행하여 분석, 출력한 것과 일치하는 것을 알 수 있다.

**3. 참고 문헌 및 정보**

- <http://ehclub.co.kr/2554>

- Laura Chappell, 와이어샤크 개론, 이재광, 전태일 옮김, 에이콘(2013), p29 – 30