PCAP Programming

28반 심수용

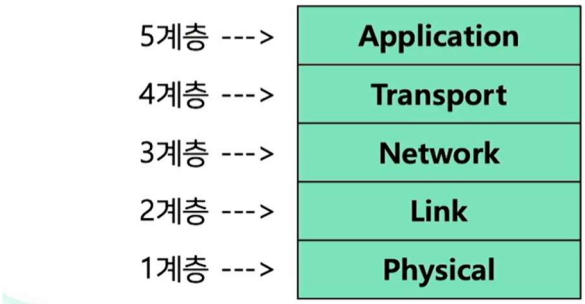
깃허브 주소: https://github.com/protruser/networkSniffing/

흔히 말하는 네트워크는 각 endpoint끼리 통신을 할 때 쓰입니다. 여기서 endpoint는 정보를 주고받을 때, 최초 송신자와 최종 수신자에 해당합니다. 프로토콜을 통해 정보를 전달하는 switching의 종류에는 2가지가 존재합니다. 각 2가지는 circuit switching과 packet switching이 있습니다. circuit switching의 경우, 고정된 회선을 통해 네트워크를 점유하여 연속적으로 데이터를 주고받기 때문에 데이터가 순차적으로 흐를 수 있습니다. 반면 packet switching의 경우에는 endpoint 기기 수에 비해 대역폭은 현저히 적기 때문에, 네트워크를 패킷 단위로 나누어서 수신자 endpoint에 각각 전달합니다. 따라서 네트워크를 들여다보면 송신자가 보낸 패킷 정보를 알 수 있고, 이것을 Sniffing이라고 합니다.

Protocol

프로토콜은 5계층 구조를 가지고 있습니다. 각 계층은 Application 계층, Transport 계층, Network 계층, Link 계층 그리고 Physical 계층을 나타냅니다. 데이터가 전송될 때, 정보는 아래의 5계층을 거치게 됩니다. 패킷이 흩어지게 되는 packet switching 특성상 재조립이 필요하므로, 각 정보를 담을 수 있는 수단이 필요하며 이것을 header라고 합니다. 정보는 각 계층을 지날 때 마다 encapsulation을 통해 헤더 정보가 추가가 됩니다. 각 계층마다 헤더가 차례로 쌓이게 되며, link 계층에서 최종적으로 헤더 정보가 모두 쌓인 패킷을 얻을 수 있습니다. 해당 패킷을 스니핑하고 각 패킷의 Mac 주소, IP 주소, port 번호

그리고 가능하다면 데이터까지 출력하는 것이 이번 프로젝트의 목표입니다.



데이터를 전송하는 경로 상에서 패킷을 스니핑하여 원하는 정보를 얻을 수 있고, 해당 코드는 C언어 그리고 PCAP API를 이용하여 작성할 수 있습니다. 코드는 다음과 같습니다.

```
#include <stdlib.h>
#include <stdio.h>
#include <pcap.h>
#include <arpa/inet.h>
```

먼저 헤더파일은 총 4개로 선언하였습니다.

- stdlib: 메모리 할당과 관련된 라이브러리
- stdio: 표준 입출력 라이브러리
- pcap: 네트워크 패킷을 캡쳐하고 분석할 수 있는 라이브러리
- arpa/inet: ip 주소 처리와 관련된 라이브러리

```
/* Ethernet header */
struct Ethernet_header {
        u_char ether_dhost[6]; // destination host address
        u_char ether_shost[6]; // source host address
        u_short ether_type;    // protocol type(IP, ARP, RARP ...)
};
```

그 다음으로 Ethernet header 구조체를 선언하였습니다.

- ether_dhost: 시작점의 mac 주소
- ether_shost: 도착점의 mac 주소

- ether_type: 프로토콜 타입

mac 주소는 6바이트 형식으로 구성되며 xx:xx:xx:xx:xx:xx 형식으로 나타납니다.

```
/* IP header */
struct Ip_header {
        unsigned char iph_ihl:4, // ip header length
                      iph_ver:4; // ip version
        unsigned char iph_tos; // type of service
        unsigned short int iph_len; // ip packet length(data + header)
        unsigned short int iph_ident; // identification
        unsigned short int iph_flag:3, // fragmentation flags
                       iph_offset:13; // flags offset
        unsigned char iph_ttl; // time to live
        unsigned char iph_protocol; // protocol type
        unsigned short int iph_chksum; // ip datagram checksum
        struct in_addr iph_src; // source IP address
        struct in_addr iph_dst; // destination IP address
};
```

이후로 Ip header 구조체를 선언하였습니다.

- iph_ihl: ip의 길이
- iph_len: ip 헤더의 길이
- iph_protocol: transport 계층에서 사용되는 프로토콜
- iph_src: 시작점의 ip 주소
- iph_dst: 도착점의 ip 주소

```
/* TCP header */
struct Tcp_header {
    unsigned short tcph_sport;      // source tcp port
    unsigned short tcph_dport;      // destination tcp port
    unsigned int tcph_seqNum;       // tcp sequence number
    unsigned int tcph_ackNum;       // tcp ack number
    unsigned char tcph_offsetx2:4;  // tcp offset
    unsigned char tcph_reversed:4;  // reversed bits
    unsigned char tcph_flags;       // tcp flags
    unsigned short tcph_window;     // window size
    unsigned short tcph_chksum;     // checksum
    unsigned short tcph_urgp;       // urgent pointer
};
```

마지막으로 Tcp header 구조체를 선언했습니다.

- tcph_sport: 시작점의 tcp 포트번호
- tcph_dport: 도착점의 tcp 포트번호

- tcp_offsetx2: tcp 오프셋 값

```
/* got_packing function */
void got_packet(unsigned char *args, const struct pcap_pkthdr *header, const unsigned char *packet)
{
        struct Ethernet_header *eth = (struct Ethernet_header *) packet;

        /* Query about Ip, Tcp and Message */
        if (ntohs(eth -> ether_type) == 0x0800) { // if packet is Ipv4


                // Ip address
                struct Ip_header *ip = (struct Ip_header *) (packet + sizeof(struct Ethernet_header));

                // Only Tcp address
                if (ip -> iph_protocol == IPPROTO_TCP) {

                        printf("============================================\n");
                        // Mac address(from)
                        printf("Ethernet From: ");
                        for (int i = 0; i < 6; i++) {
                                printf("%02x", eth -> ether_shost[i]);
                                if (i == 5) break;
                                printf(":");
                        }
                        printf("\n");


                        // Mac address(to)
                        printf("Mac To: ");
                        for (int i = 0; i < 6; i++) {
                                printf("%02x", eth -> ether_dhost[i]);
                                if (i == 5) break;
                                printf(":");
                        }
                        printf("\n");
                        printf("\n");


                        // Ip address
                        int ip_packet_len = ntohs(ip -> iph_len);

                        printf("IP From: %s\n",inet_ntoa(ip -> iph_src));
                        printf("IP To: %s\n", inet_ntoa(ip -> iph_dst));
                        printf("\n");
```

```
                        int ip_header_len  = ip -> iph_ihl * 4;
                        struct Tcp_header *tcp = (struct Tcp_header *) (packet + sizeof(struct Ethernet_header) + ip_header_len);
                        printf("TCP From: %d\n", ntohs(tcp -> tcph_sport));
                        printf("TCP To: %d\n", ntohs(tcp -> tcph_dport));
                        printf("\n");

                        int tcp_header_len = (tcp-> tcph_offsetx2) * 4;

                        // Message
                        unsigned char *msg = (unsigned char *)(packet + sizeof(struct Ethernet_header) + ip_header_len + tcp_header_len);
                        unsigned int length = ip_packet_len - ip_header_len - tcp_header_len;

                        printf("Message: ");
                        for(int i = 0; i < length; i++) {
                                printf("%c", msg[i]);
                        }

                        printf("\n");
                        printf("============================================\n");
                }

        }
}
```

그리고 패킷을 스니핑하는 함수는 위와 같습니다. 하나하나씩 뜯어 보겠습니다.

```
/* got_packing function */
void got_packet(unsigned char *args, const struct pcap_pkthdr *header, const unsigned char *packet)
{
        struct Ethernet_header *eth = (struct Ethernet_header *) packet;

        /* Query about Ip, Tcp and Message */
        if (ntohs(eth -> ether_type) == 0x0800) { // if packet is Ipv4
```

got_packet은 함수가 패킷을 캡쳐할 때마다 호출되는 콜백 함수입니다. 각 함수의 파라

미터 정보는 사용자 정의 데이터, 메타데이터, 패킷 데이터입니다. 마지막 파라미터가 패킷 데이터이므로 이 파라미터를 통해 패킷 정보를 하나하나씩 뜯어볼 수 있습니다.

먼저 Ethernet_header 구조체와 연관된 변수를 선언하고 파라미터로 얻은 패킷을 형변환 시킵니다. 그리고 ntohs 함수를 이용하여 ether_type이 Ipv4에 해당하는 프로토콜만으로 조건을 걸었습니다.

```
// Ip address
struct Ip_header *ip = (struct Ip_header *) (packet + sizeof(struct Ethernet_header));

// Only Tcp address
if (ip -> iph_protocol == IPPROTO_TCP) {

        printf("=============================================\n");
        // Mac address(from)
        printf("Ethernet From: ");
        for (int i = 0; i < 6; i++) {
                printf("%02x", eth -> ether_shost[i]);
                if (i == 5) break;
                printf(":");
        }
        printf("\n");


        // Mac address(to)
        printf("Mac To: ");
        for (int i = 0; i < 6; i++) {
                printf("%02x", eth -> ether_dhost[i]);
                if (i == 5) break;
                printf(":");
        }
        printf("\n");
        printf("\n");
```

이후 Ip_header 구조체와 연관된 ip 변수를 선언하고 패킷에서 ethernet_header 분리시킵니다. 이후 iph_protocol 변수를 통해 TCP 프로토콜에 해당하는 패킷만을 조건으로 출력합니다. 그리고 이전에 받아두었던 eth 변수를 통해 시작점의 mac 주소와 도착점의 mac 주소를 출력합니다.

```
// Ip address
int ip_packet_len = ntohs(ip -> iph_len);

printf("IP From: %s\n",inet_ntoa(ip -> iph_src));
printf("IP To: %s\n", inet_ntoa(ip -> iph_dst));
printf("\n");
```

Ip packet 길이를 ip_packet_len에 담고, mac 주소 출력과 동일하게 IP 주소도 출력을 진행합니다.

```
int ip_header_len  = ip -> iph_ihl * 4;
struct Tcp_header *tcp = (struct Tcp_header *) (packet + sizeof(struct Ethernet_header) + ip_header_len);
printf("TCP From: %d\n", ntohs(tcp -> tcph_sport));
printf("TCP To: %d\n", ntohs(tcp -> tcph_dport));
printf("\n");

int tcp_header_len = (tcp-> tcph_offsetx2) * 4;

// Message
unsigned char *msg = (unsigned char *)(packet + sizeof(struct Ethernet_header) + ip_header_len + tcp_header_len);
unsigned int length = ip_packet_len - ip_header_len - tcp_header_len;

printf("Message: ");
for(int i = 0; i < length; i++) {
        printf("%c", msg[i]);
}

printf("\n");
printf("=============================================\n");
```

현장 강의 자료를 바탕으로 가변적인 ip 헤더 길이를 출력하고 ip_header_len에 저장합니다.

이후 Tcp_header 구조체 변수로 tcp를 만들고 각각 패킷을 길이와 함께 분리합니다. 분리를 진행한 후 Tcp의 시작점 port와 도착점 port를 출력합니다.

그리고 Tcp header에 있는 data offset을 가져오기 위해 data offset 기준 4를 곱해서 tcp 헤더 길이를 구했습니다.

이후 message에서 tcp 길이까지 분리를 진행하였고, 최종적으로 data만 남겨 출력을 진행하였습니다.

```
int main()
{
        pcap_t *handle;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct bpf_program fp;
        char filter_exp[] = "tcp port 80";
        bpf_u_int32 net = 0;

        handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

        pcap_compile(handle, &fp, filter_exp, 0, net);
        if (pcap_setfilter(handle, &fp) != 0) {
                pcap_perror(handle, "Error:");
                exit(EXIT_FAILURE);
        }

        pcap_loop(handle, -1, got_packet, NULL);

        pcap_close(handle);
        return 0;
}
```

해당 main 함수는 수업시간에 진행한 sniff_improved.c 코드와 동일하게 진행하였습니다.

다만 컴파일 과정에서 net 변수가 0으로 제대로 초기화가 되지 않은 문제가 있어 0으로 수동 기입 하였습니다.

최종 실행 코드 결과

```
user1@SimSuyong:~/network_security_codes/Sniffing_Spoofing$ sudo ./network_sniffing
==========================================
Ethernet From: 08:00:27:3b:75:55
Mac To: 52:54:00:12:35:02

IP From: 10.0.2.15
IP To: 34.107.221.82

TCP From: 58050
TCP To: 80

Message: ◆◆P3T◆T◆◆◆
                  ◆◆
6◆◆◆
==========================================
```

파이어폭스를 열었을 당시에 mac주소, ip 주소, port 번호가 정상적으로 출력되는 것을 확인했습니다. 또한, message는 암호화된 형태로 패킷이 송신된 것을 확인했습니다.

```
==========================================
Ethernet From: 52:54:00:12:35:02
Mac To: 08:00:27:3b:75:55

IP From: 23.192.228.80
IP To: 10.0.2.15

TCP From: 80
TCP To: 36280

Message: P◆◆(◆>P;n'P◆◆◆s domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>

==========================================
```

이번에는(http://example.com)으로 접속하여, 패킷 스니핑을 진행했습니다.

패킷 중 특정 데이터는 평문으로 출력되는 것을 확인했습니다. 하지만 정확한 데이터 시작점을 찾는 부분에는 어려움이 있어서 일부 메시지가 덤프된 부분이 존재하였습니다.