칼리 리눅스(Kali Linux) 64비트 운영체제
PCAP 기초 프로그래밍 메뉴얼

작성자: 나동빈

※ PCAP 개요 ※

PCAP(Packet Capture)은 네트워크상의 패킷을 캡처하고 관리하기 위한 라이브러리입니다.

※ PCAP 기초 프로그래밍 ※

(PCAP 라이브러리 공식 홈페이지 http://www.tcpdump.org/에서 더욱 자세히 설명되어 있습니다.)

```
#include <stdio.h>
#include <pcap.h> // PCAP 라이브러리 가져오기
#include <arpa/inet.h> // inet_ntoa 등 함수 포함
#include <netinet/in.h> // in_addr 등 구조체 포함
```

먼저 위와 같이 필요한 라이브러리를 추가해줍니다.

```
pcap_t *handle; // 핸들러
char *dev = "eth0"; // 자신의 네트워크 장비
char errbuf[PCAP_ERRBUF_SIZE]; // 오류 메시지를 저장하는 버퍼
struct bpf_program fp; // 필터 구조체
char *filter_exp = "port 80"; // 필터 표현식
bpf_u_int32 mask; // 서브넷 마스크
bpf_u_int32 net; // 아이피 주소
struct pcap_pkthdr *header; // 패킷 관련 정보
const u_char *packet; // 실제 패킷
struct in_addr addr; // 주소 정보
```

PCAP 라이브러리를 사용할 때의 기본적인 전역변수 형태입니다.

소스코드	설명
pcap_t *handle;	우리의 네트워크 장비와 통신하며 세션을 관리해주는 변수를 선언합니다.
char *dev = "eth0";	자신의 네트워크 장비를 입력합니다. 칼리 리눅스는 eth0입니다.
errbuf[PCAP_ERRBUF_SIZE];	오류가 발생하면 오류 메시지를 저장하고 이후에 출력할 수 있도록 합니다.
struct bpf_program fp;	필터 구조체입니다. 컴파일되어 사용가능한 형태가 됩니다.
char *filter_exp = “port 80";	실제로 필터 내용을 담습니다. ”port 80“이란 80번 포트(HTTP)만 캡처하겠다는 의미입니다. 결과적으로 특정 웹 사이트에 접속했을 때 주고받는 패킷만 캡처가 이루어집니다.
bpf_u_int32 mask;	서브넷 마스크를 담을 변수를 선언합니다.
bpf_u_int32 net;	아이피 주소를 담을 변수를 선언합니다.
struct pcap_pkthdr header;	캡처된 패킷과 관련한 기타 정보가 저장되는 변수를 선언합니다.
const u_char *packet;	**실제로 캡처된 패킷이 담기는 변수**입니다.
struct in_addr addr;	자신의 주소 정보를 저장하는 변수입니다.

이후에 메인 함수를 작성합니다.

```
int main(void) {
        dev = pcap_lookupdev(errbuf);
        if (dev == NULL) {
                printf("네트워크 장치를 찾을 수 없습니다.₩n");
                return 0;
        }
        printf("나의 네트워크 장치: %s₩n", dev);
        if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
                printf("장치의 주소를 찾을 수 없습니다.₩n");
                return 0;
        }
        addr.s_addr = net;
        printf("나의 IP주소: %s₩n", inet_ntoa(addr));
        addr.s_addr = mask;
        printf("나의 서브넷 마스크: %s₩n", inet_ntoa(addr));
        return 0;
}
```

소스코드	설명
pcap_lookupdev();	자동으로 자신의 네트워크 장치를 탐색합니다.
pcap_lookupnet();	자신의 장치에 대한 IP 주소와 서브넷 마스크 주소를 가져옵니다.
inet_ntoa();	네트워크 32비트 값을 IP 주소 형태(255.255.255.255)로 변환합니다.

```
#include <stdio.h>
#include <pcap.h> // PCAP 라이브러리 가져오기
#include <arpa/inet.h> // inet_ntoa 등 함수 포함
#include <netinet/in.h> // in_addr 등 구조체 포함

pcap_t *handle; // 핸들러
char *dev = "eth0"; // 자신의 네트워크 장비
char errbuf[PCAP_ERRBUF_SIZE]; // 오류 메시지를 저장하는 버퍼
struct bpf_program fp; // 필터 구조체
char *filter_exp = "port 80"; // 필터 표현식
bpf_u_int32 mask; // 서브넷 마스크
bpf_u_int32 net; // 아이피 주소
struct pcap_pkthdr header; // 패킷 관련 정보
const u_char *packet; // 실제 패킷
struct in_addr addr; // 주소 정보

int main(void) {
        dev = pcap_lookupdev(errbuf);
        if (dev == NULL) {
                printf("네트워크 장치를 찾을 수 없습니다.₩n");
                return 0;
        }
        printf("나의 네트워크 장치: %s₩n", dev);
        if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
                printf("장치의 주소를 찾을 수 없습니다.₩n");
                return 0;
        }
        addr.s_addr = net;
        printf("나의 IP주소: %s₩n", inet_ntoa(addr));
        addr.s_addr = mask;
        printf("나의 서브넷 마스크: %s₩n", inet_ntoa(addr));
        return 0;
}
```

위와 같이 전체 소스코드를 구성하게 된 것입니다. 이제 실행하면 다음과 같습니다.

```
root@KaliLinux:~/libpcap-1.8.1# nano example.c
root@KaliLinux:~/libpcap-1.8.1# gcc -o example example.c -lpcap
root@KaliLinux:~/libpcap-1.8.1# ./example
나의 네트워크 장치: eth0
나의 IP주소: 192.168.32.0
나의 서브넷 마스크: 255.255.255.0
```

이제 특정한 포트를 거쳐 실제로 패킷을 캡처하겠습니다.

```
handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
        printf("장치를 열 수 없습니다.₩n");
        return 0;
}
if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
        printf("필터를 적용할 수 없습니다.₩n");
        return 0;
}
if (pcap_setfilter(handle, &fp) == -1) {
        printf("필터를 세팅할 수 없습니다.₩n");
        return 0;
}
```

위와 같이 작성합니다.

소스코드	설명
pcap_open_live()	패킷 스니핑 세션을 생성합니다. 지금의 경우 한 번 캡처하는 최대 바이트 크기를 BUFSIZ, 패킷 캡처 시간이 1,000ms(1초)로 설정되었습니다.
pcap_compile()	해당 세션에 패킷 스니핑 필터를 적용하기 위해 컴파일을 수행합니다.
pcap_setfilter()	실제로 필터링을 수행합니다.

```
#define ETHER_ADDR_LEN 6

struct sniff_ethernet {
        u_char ether_dhost[ETHER_ADDR_LEN]; // 목적지 MAC 주소
        u_char ether_shost[ETHER_ADDR_LEN]; // 출발지 MAC 주소
        u_short ether_type;
};
```

이제 위와 같이 이더넷(Ethernet) 스니핑 구조체를 선언합니다. 실제로 사용할 것은 목적지, 출발지 주소입니다.

```
#define IP_HL(ip)       (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)        (((ip)->ip_vhl) >> 4)

struct sniff_ip {
        u_char  ip_vhl;
        u_char  ip_tos;
        u_short ip_len;
        u_short ip_id;
        u_short ip_off;
        #define IP_RF 0x8000
        #define IP_DF 0x4000
        #define IP_MF 0x2000
        #define IP_OFFMASK 0x1fff
        u_char  ip_ttl;
        u_char  ip_p;  // IP 프로토콜 유형
        u_short ip_sum;
        struct in_addr ip_src; // 출발지 IP 주소
        struct in_addr ip_dst; // 목적지 IP 주소
};
```

이후에 위와 같이 IP 스니핑 구조체를 선언합니다. 프로토콜 유형, 출발지, 목적지 주소를 사용하겠습니다.

```
typedef u_int tcp_seq;

struct sniff_tcp {
        u_short th_sport; // 출발지 TCP 주소
        u_short th_dport; // 목적지 TCP 주소
        tcp_seq th_seq;
        tcp_seq th_ack;
        u_char th_offx2;
        #define TH_OFF(th) (((th)->th_offx2 & 0xf0) >> 4)
        u_char th_flags;
        #define TH_FIN 0x01
        #define TH_SYN 0x02
        #define TH_RST 0x04
        #define TH_PUSH 0x08
        #define TH_ACK 0x10
        #define TH_URG 0x20
        #define TH_ECE 0x40
        #define TH_CWR 0x80
        #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;
        u_short th_sum;
        u_short th_urp;
};
```

그리고 TCP 스니핑 구조체도 선언합니다. 이것도 목적지, 출발지 주소만 사용하겠습니다. 보면 전부 다 목적지, 출발지 주소만 사용하는데 이를 전체 16진수 형태의 패킷을 직접 쪼개서 따낼 수도 있습니다. 하지만 위와 같이 구조체로 선언해서 만들게 되면 알아서 구조체 내부 변수로 크기가 분리되므로 더 쉽게 패킷을 따낼 수 있는 것입니다.

```
#define SIZE_ETHERNET 14

struct sniff_ethernet *ethernet; // 이더넷 헤더
struct sniff_ip *ip; // IP 헤더
struct sniff_tcp *tcp; // TCP 헤더
char *payload; // 페이로드

u_int size_ip;
u_int size_tcp;
```

이제 위와 같이 변수를 초기화해서 사용해보도록 하겠습니다.

```
while(pcap_next_ex(handle, &header, &packet) >= 0) {
        parsing();
}
```

이제 위와 같이 메인 함수에 넣어줍니다. pcap_next_ex()는 캡처된 패킷 내용을 packet 변수에 담습니다. 성공적으로 오류가 발생하지 않는 한에서 패킷을 반복적으로 캡처해서 parsing()이라는 함수를 불러옵니다. pcap_next_ex()는 성공적으로 수행이 되면 1을 반환합니다. 시간초과가 발생하면 0을 반환합니다. 그리고 패킷을 읽는 도중에 오류가 발생한 경우 -1을 반환합니다. 이제 parsing() 함수를 작성해주면 됩니다.

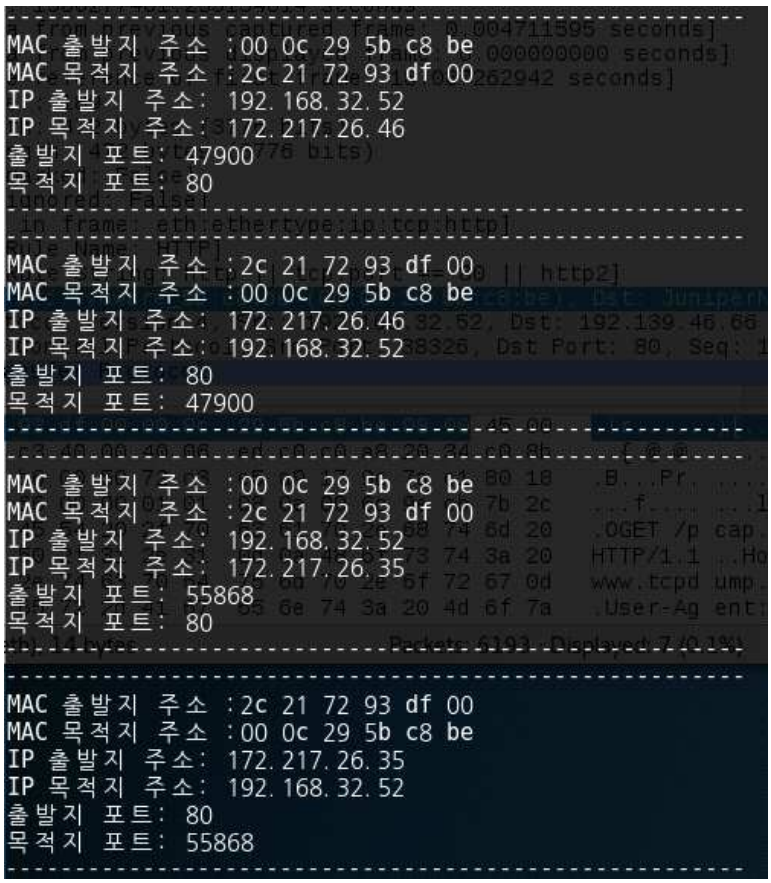
```
void parsing() {
        printf("-------------------------------------------------------₩n");
        int i;
        ethernet = (struct sniff_ethernet*)(packet);
        printf("MAC 출발지 주소 :");
        for(i = 0; i < ETHER_ADDR_LEN; i++) {
                printf("%02x ", ethernet->ether_shost[i]);
        }
        printf("₩nMAC 목적지 주소 :");
        for(i = 0; i < ETHER_ADDR_LEN; i++) {
                printf("%02x ", ethernet->ether_dhost[i]);
        }
        ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
        size_ip = IP_HL(ip)*4;
        printf("₩nIP 출발지 주소: %s₩n", inet_ntoa(ip->ip_src));
        printf("IP 목적지 주소: %s₩n", inet_ntoa(ip->ip_dst));
        tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
        size_tcp = TH_OFF(tcp)*4;
        printf("출발지 포트: %d₩n", ntohs(tcp->th_sport));
        printf("목적지 포트: %d₩n", ntohs(tcp->th_dport));
        printf("-------------------------------------------------------₩n");
}
```

위와 같이 작성해봅시다.

소스코드	설명
ethernet = (struct sniff_ethernet*)(packet);	패킷의 헤더 첫 번째 부분은 MAC 주소의 정보를 보여줍니다. 따라서 바로 ethernet 포인터 변수가 패킷 변수의 주소를 가리키도록 합니다.
ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);	이더넷 정보를 나타내는 부분은 총 길이가 14바이트입니다. 그 뒤에는 바로 IP 헤더가 잇습니다.
inet_ntoa()	비트 형태의 IP 주소를 문자열로 바꿉니다.
tcp = (struct sniff_tcp*) (packet + SIZE_ETHERNET + size_ip);	IP 헤더가 끝나고 난 뒤에는 TCP 헤더가 있습니다.
ntohs()	비트 형태의 포트 번호를 정수형으로 바꿉니다.

```
root@KaliLinux:~/libpcap-1.8.1# ./example
나의 네트워크 장치: eth0
나의 IP주소: 192.168.32.0
나의 서브넷 마스크: 255.255.255.0
패킷을 감지합니다.
```

이제 실행해보면 위와 같이 패킷 감지가 시작됩니다. 현재 포트 번호를 80으로 잡았기 때문에 HTTP 프로토콜(WEB)에 한해서 모든 패킷들을 확인해 볼 수 있습니다. 따라서 이제 HTTP를 사용하는 웹 사이트에 접속합니다.



이제 웹 사이트에 접속하면 위와 같이 패킷이 캡처가 됩니다. 이제 구체적인 **페이로드(Payload)**를 출력해볼 수 있도록 합시다. 페이로드는 각종 주소 헤더 정보를 제외하고, 근본적으로 주고받으려는 데이터를 의미합니다.

```
void parsing() {
        printf("------------------------------------------------------\n");
        int i, payload_len;
        ethernet = (struct sniff_ethernet*)(packet);
        printf("MAC 출발지 주소 :");
        for(i = 0; i < ETHER_ADDR_LEN; i++) {
                printf("%02x ", ethernet->ether_shost[i]);
        }
        printf("\nMAC 목적지 주소 :");
        for(i = 0; i < ETHER_ADDR_LEN; i++) {
                printf("%02x ", ethernet->ether_dhost[i]);
        }
        ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
        size_ip = IP_HL(ip)*4;
        printf("\nIP 출발지 주소: %s\n", inet_ntoa(ip->ip_src));
        printf("IP 목적지 주소: %s\n", inet_ntoa(ip->ip_dst));
        tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
        size_tcp = TH_OFF(tcp)*4;
        printf("출발지 포트: %d\n", ntohs(tcp->th_sport));
        printf("목적지 포트: %d\n", ntohs(tcp->th_dport));
        payload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp);
        payload_len = ntohs(ip->ip_len) - (size_ip + size_tcp);
        if(payload_len == 0) printf("페이로드 데이터가 없습니다.");
        else {
                printf("< 페이로드 데이터 >\n");
                for(int i = 1; i < payload_len; i++) {
                        printf("%02x ", payload[i - 1]);
                        if(i %8 == 0) printf("  ");
                        if(i %16 == 0) printf("\n");
                }
        }
        printf("\n------------------------------------------------------\n");
}
```

이제 위와 같이 parsing() 함수를 작성하면 프로그램 작성이 완료됩니다. 기존의 코드에 페이로드를 출력하는 부분만 덧붙여 넣은 것입니다. 그 코드에 대한 내용은 아래와 같습니다.

소스코드	설명
payload = (u_char*) (packet + SIZE_ETHERNET + size_ip + size_tcp);	페이로드가 존재하는 패킷 부분은 이더넷 부분, IP 부분, TCP 부분의 다음입니다.
payload_len = ntohs(ip->ip_len) - (size_ip + size_tcp)	페이로드의 전체 길이를 계산합니다.

for(int I = 1; I < payload.len; I++) {
printf("%02x ", payload[i - 1]);
if(i % 8 == 0) printf(" ");
if(i % 16 == 0) printf("₩n");
}
	페이로드를 출력합니다. 1바이트씩 8개를 차례대로 두 번 출력하고 한 번 줄 바꿈을 하는 방식으로 사용자가 보기 편하게 만듭니다.