

pcap-test 과제

▼ 유형	과제
📎 자료	https://gitlab.com/gilgil/sns/-/wikis/pcap-programming/report-pcap-test
☑ 복습 여부	<input type="checkbox"/>

Report pcap test

과제

실행

상세

Pcap-test 과제 해석

1.1. 원리

1.2. skeleton 코드 해석

1.3. pcap-test code 해석

Report pcap test

과제

송수신되는 packet을 capture하여 중요 정보를 출력하는 C/C++ 기반 프로그램을 작성하라.



조건

1. Ethernet Header의 src mac / dst mac
2. IP Header의 src ip / dst ip
3. TCP Header의 src port / dst port
4. Payload(Data)의 hexadecimal value(최대 10바이트까지만)

실행

```
syntax: pcap-test <interface>
sample: pcap-test wlan0
```

상세

- TCP packet이 잡히는 경우 "ETH + IP + TCP + DATA" 로 구성이 된다. 이 경우(TCP packet이 잡혔다고 판단되는 경우만)에만 1~4의 정보를 출력하도록 한다(Data의 크기가 0여도 출력한다).
- 각각의 Header에 있는 특정 정보들(mac, ip, port)를 출력할 때, 노다가(packet의 시작위치로부터 일일이 바이트 세어 가며)로 출력해도 되는데 불편함.
- 이럴 때 각각의 Header 정보들이 구조체로 잘 선언한 파일이 있으면 코드의 구성이 한결 간결해진다. 앞으로 가급적이면 네트워크 관련 코드를 작성할 할 때에는 libnet 혹은 자체적인 구조체를 선언하여 사용하도록 한다.
 - <http://packetfactory.openwall.net/projects/libnet> > Latest Stable Version: 1.1.2.1 다운로드(libnet.tar.gz) > include/libnet/libnet-headers.h
 - libnet-headers.h 안에 있는 본 과제와 직접적으로 관련된 구조체들 :
 - struct libnet_ethernet_hdr (479 line)

- struct libnet_ipv4_hdr (647 line)
- struct libnet_tcp_hdr (1519 line)
- pcap-test 코드를 skeleton code으로 하여 과제를 수행해야 하며, pcap_findalldevs, pcap_compile, pcap_setfilter, pcap_lookupdev, pcap_loop API는 사용하지 않는다(인터넷에 돌아다니는 코드에 포함되어 있는 함수들이며, 본 함수들이 과제의 코드에 포함되는 경우 과제를 베낀 것으로 간주함).

Pcap-test 과제 해석

1.1. 원리

os 7 layer을 보면 통신이 L1부터 차례대로 경로에 따라서 올라가는 것을 지난 시간에 배웠다.

memory	OSI	Layer No	TCP/IP	OTHERS
Alligator	Application	L7	HTTP	HTTPS, FTP, SSH, LOCO etc..
Pet	Presentation			
Steve's	Session			
Touch	Transport	L4	TCP	UDP, SCTP, ICMP
Not	Network	L3	IP	IPv6, ARP
Do	Data Link	L2	Ethernet	Frame Relay
Please	Physical	L1		

linux 안에 wireshark로 잡은 패킷을 예시를 보게되면 TCP 통신을 한 패킷을 보게 되면,

Ethernet → IP → TCP 로 통신 했음을 알 수 있다. (대부분이 Ethernet frame을 사용하기 때문에 처음 시작하는 출발 장소)

No.	Time	Source	Destination	Protocol	Length	Info
103	17.967851761	127.0.0.1	127.0.0.1	DNS	86	Standard query 0xc7cb
104	17.967860675	127.0.0.1	127.0.0.1	ICMP	114	Destination unreachable
105	17.967899118	127.0.0.1	127.0.0.1	DNS	86	Standard query 0xc7cb
106	17.967905952	127.0.0.1	127.0.0.1	ICMP	114	Destination unreachable
107	19.792498891	127.0.0.1	127.0.0.1	TCP	76	41488 → 5037 [SYN] Seq
108	19.792518417	127.0.0.1	127.0.0.1	TCP	56	5037 → 41488 [RST, ACK
109	19.793857413	127.0.0.1	127.0.0.1	TCP	76	41490 → 5037 [SYN] Seq
110	19.793865572	127.0.0.1	127.0.0.1	TCP	56	5037 → 41490 [RST, ACK

Frame 107: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0	0000	00 00 03 04 00 06 00 00 00 00 00 00 00 06 08
Linux cooked capture v1	0010	45 00 00 3c d6 cf 40 00 40 06 65 ea 7f 00 00
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0020	7f 00 00 01 a2 10 13 ad 0f 86 ce 7a 00 00 00
Transmission Control Protocol, Src Port: 41488, Dst Port: 5037	0030	a0 02 ff d7 fe 30 00 00 02 04 ff d7 04 02 08
	0040	d6 d0 8a 21 00 00 00 00 01 03 03 07

+) Linux cooked capture v1 이란?

리눅스에서 "임의의" 장치 또는 다른 장치 중 하나에서 캡처할 때 libpcap은 이더넷과 같은 실제 "하드웨어 프로토콜"에 대한 링크 계층 헤더를 제공하지 않고 대신 이 유사 프로토콜에 대한 가짜 링크 계층 헤더를 제공한다.

IPv4 를 펼쳐보면 Src와 Dst이 localhost 라는 것을 알 수 있으며, IPv4와 같은 경우 이전 계층에서 protocol이 **0x800** 이면 **IPv4**라는 것을 알 수 있다.

```

> Frame 107: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
> Linux cooked capture v1
  Packet type: Unicast to us (0)
  Link-layer address type: Loopback (772)
  Link-layer address length: 6
  Source: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Unused: 0006
  Protocol: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 41488, Dst Port: 5037

```

```

> Frame 107: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0xd6cf (54991)
> 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0x65ea [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 127.0.0.1
  Destination Address: 127.0.0.1
> Transmission Control Protocol, Src Port: 41488, Dst Port: 5037

```

IP에 있는 protocol을 보면 **60**이 **tcp**인 것을 확인할 수 있다.

tcp를 확장해보면 Src Port가 41488 이며, Dst Port가 5037인 것을 알 수 있다.

```

Transmission Control Protocol, Src Port: 41488, Dst Port: 5037
  Source Port: 41488
  Destination Port: 5037
  [Stream index: 0]
  [Conversation completeness: Incomplete (37)]
  [TCP Segment Len: 0]
  Sequence Number: 0      (relative sequence number)
  Sequence Number (raw): 260492922
  [Next Sequence Number: 1      (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  1010 .... = Header Length: 40 bytes (10)
  Flags: 0x002 (SYN)
  Window: 65495
  [Calculated window size: 65495]
  Checksum: 0xfe30 [unverified]

```

1.2. skeleton 코드 해석

```

#include <pcap.h> // pcap 라이브러리 헤더 파일
#include <stdbool.h>
#include <stdio.h>

// 예제 출력 함수
void usage() {
    printf("syntax: pcap-test <interface>\n");
    printf("sample: pcap-test wlan0\n");
}

// 구조체
typedef struct {
    char* dev_;
} Param;

Param param = {
    .dev_ = NULL
};

bool parse(Param* param, int argc, char* argv[]) {
    if (argc != 2) { // 예외처리로 인자가 != 2 인 경우 false로 리턴
        usage();
        return false;
    }
    param->dev_ = argv[1]; // 인자로 받은 네트워크 인터페이스 이름을 변수에 저장
    return true; // true 반환
}

int main(int argc, char* argv[]) {
    if (!parse(&param, argc, argv)) // 프로그램 인자 처리 함수를 통해 인터페이스 이름이 제대로 들어오지 않은 경우
        return -1;

    char errbuf[PCAP_ERRBUF_SIZE]; // 오류 메시지를 저장할 버퍼 생성
    pcap_t* pcap = pcap_open_live(param.dev_, BUFSIZ, 1, 1000, errbuf); // 지정된 네트워크 인터페이스를 모니터링할 패킷 캡처 핸들 생성
    if (pcap == NULL) { // 패킷 캡처 핸들 생성에 실패한 경우
        fprintf(stderr, "pcap_open_live(%s) return null - %s\n", param.dev_, errbuf); // 오류 메시지 출력
        return -1; // -1 반환
    }

    while (true) { // 무한 루프
        struct pcap_pkthdr* header; // 패킷 헤더를 저장할 변수
        const u_char* packet; // 실제 패킷 데이터를 저장할 변수
        int res = pcap_next_ex(pcap, &header, &packet); // 다음 패킷을 읽어와서 저장
        if (res == 0) continue; // 패킷이 존재하지 않는 경우 루프 다시 시작
        if (res == PCAP_ERROR || res == PCAP_ERROR_BREAK) { // 패킷 캡처 도중 에러가 발생한 경우
            printf("pcap_next_ex return %d(%s)\n", res, pcap_geterr(pcap)); // 오류 메시지 출력 후 break
            break;
        }
    }
}

```

```

    }
    printf("%u bytes captured\n", header->caplen); // 패킷 길이 출력
}

pcap_close(pcap); // 패킷 캡처 핸들 해제
}

```

결과적으로 root에서 ifconfig로 eth0, wlan0 등을 날리면 패킷을 잡아서 잡은 패킷의 길이를 출력하는 코드이다.

1.3. pcap-test code 해석

결과적으로 tcp 헤더가 잡힌 경우, 최대 아래 4가지의 정보를 출력하도록 하는 코드를 짜야하는 과제이다.

1. Ethernet Header의 src mac / dst mac
2. IP Header의 src ip / dst ip
3. TCP Header의 src port / dst port
4. Payload(Data)의 hexadecimal value(최대 10바이트까지만)



과제 아이디어

1. 각각 Ethernet, IP, TCP Header을 담는 구조체를 생성한다.

→ 과제 세부사항에 주어진 libnet-headers.h 안에 있는 본 과제와 직접적으로 관련된 구조체들 :

- struct libnet_ethernet_hdr (479 line)
- struct libnet_ipv4_hdr (647 line)
- struct libnet_tcp_hdr (1519 line)

를 이용하여 생성하고자 한다.

2. 함수를 선언하여 bytepayload가 최대 10바이트만 출력되게 구현한다.

3. IP, TCP Header의 Protocol 을 활용해서 패킷들 중에 eth 에서 protocol 값이 0x800(IP), 0x86dd(IPv6) 인 경우 다음 조건으로 넘어가고 IP protocol 값에서 6(TCP)인 경우 tcp header 구조체를 불러서 src port/ dst port 를 출력하는 이중 조건문을 while 문을 활용한 무한 루프로 진행하고자 한다.