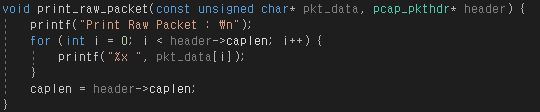
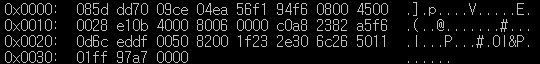
1) Make a packet sniffer using winpcap. Let it dump the raw byte stream of the packet.

Compare it with the output of windump.



windump



my sniffer



내 sniffer에 print\_raw\_packet을 정의해주고 windump와 내 sniffer를 동시에 실행시킨 뒤 인하대 홈페이지를 열었더니 windump와 내 sniffer의 결과가 똑같이 나왔다.

2) Improve your sniffer such that it also prints the header information as follows:

dest MAC: ......

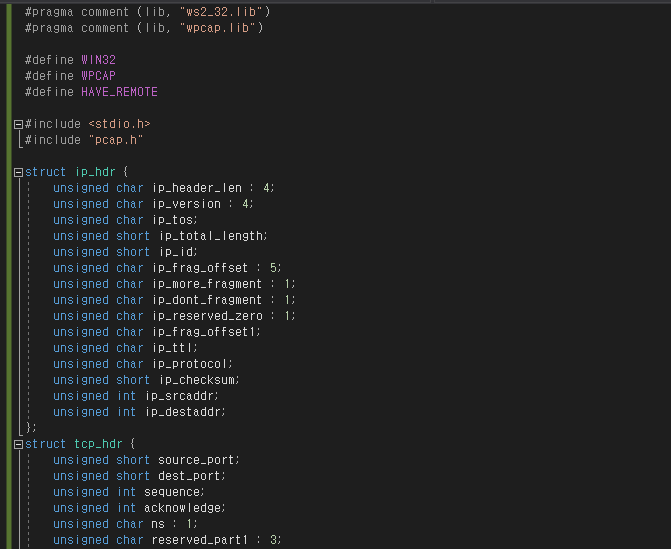
src MAC: .......

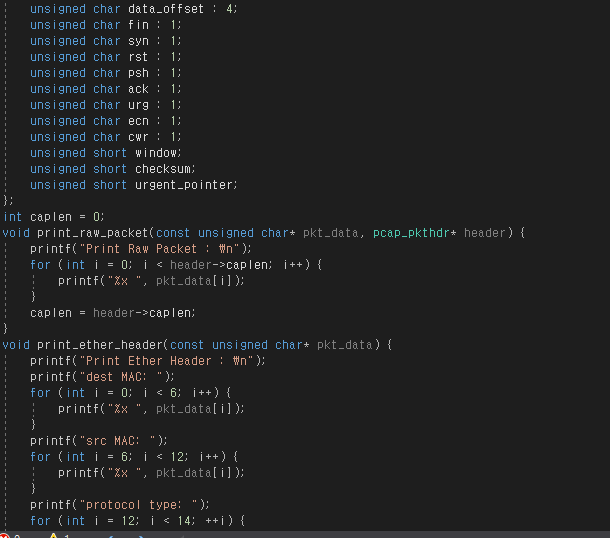
protocol type: ........

IP version: ...

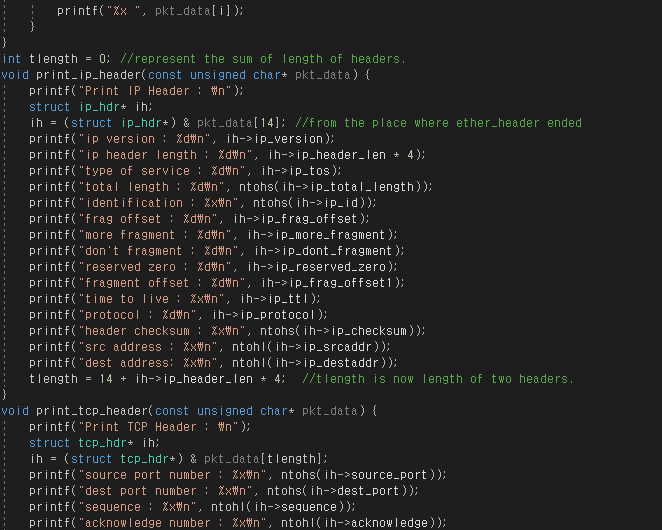
IP header length: .....

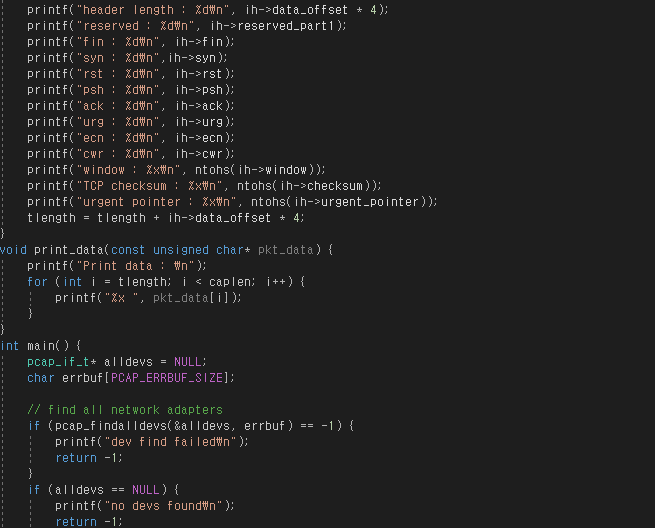
......................





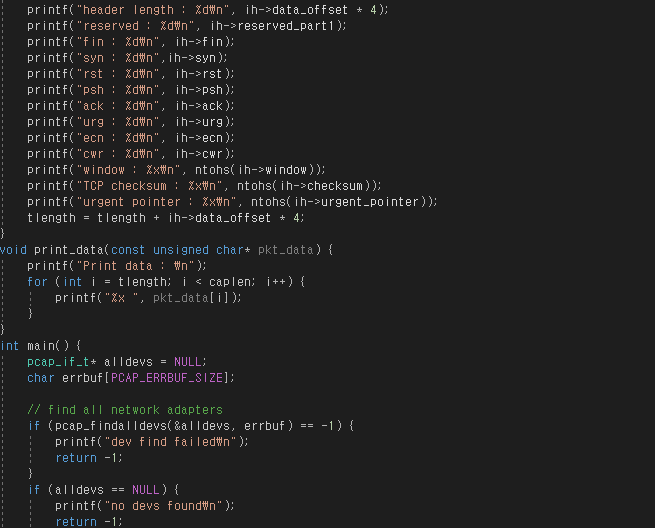
여기에서 caplen 변수는 나중에 print\_data 함수에서 사용하기 위해 만들었다. print\_raw\_packet 함수에서 pkt\_data의 총 길이인 header->caplen의 값을 caplen에 대입해줬다.



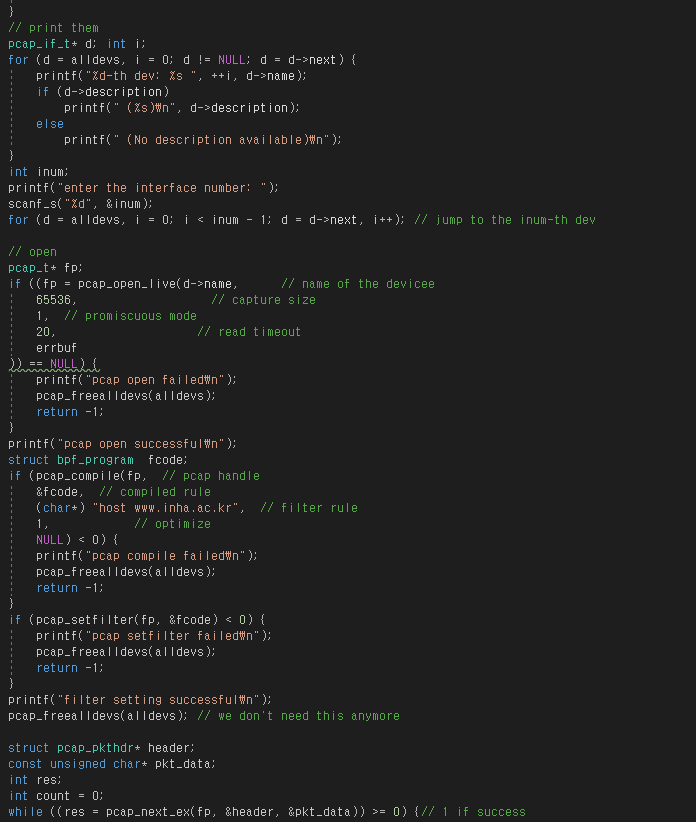


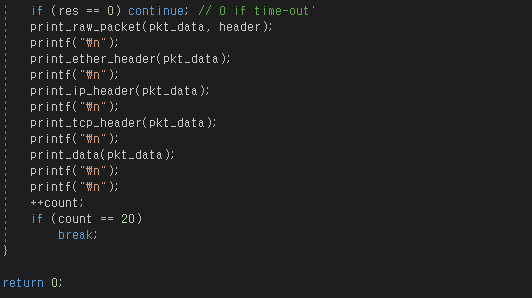
tlength 변수는 헤더들의 길이를 차례대로 더해서 다음 헤더에서 출력할 데이터의 시작위치를 표현하기 위해 만들었다. ip 헤더와 tcp 헤더는 다른 것들과 다르게 pkt\_data가 아닌 필드를 출력해야하기 때문에 array인 pkt\_data를 그대로 출력해서는 안되고 ih라는 포인터를 만들어줘서 헤더의 시작부분을 가르키도록 만들어줘야한다. 그러기 위해서 이 전에 ip\_hdr와 tcp\_hdr을 struct 해줬다. ip 헤더의 시작은 14번째부터이기 때문에 &pkt\_data[14] 라고 선언해줬다. (&는 주소값을 가리킨다.) pkt\_data는 unsigned char \* 이기 때문에 ip\_hdr의 필드를 출력해주기 위해선 type casting이 필요해서 &pkt\_data[14] 앞에 (struct ip\_hdr\*)를 넣어주었다.

출력해주는 데에는 little endian 방식이 있고 big endian 방식이 있는데 패킷 데이터는 원래 big endian 형식으로 출력하게 돼있다. 하지만 내가 원하는대로 출력되게 만들기 위해선 반드시 small endian 형식으로 출력돼야하기 때문에 2바이트인 값에는 ntohs 함수를 쓰고 4바이트인 값에는 ntohl을 써서 모든게 다 small endian 형식으로 출력되게 만들어줬다.

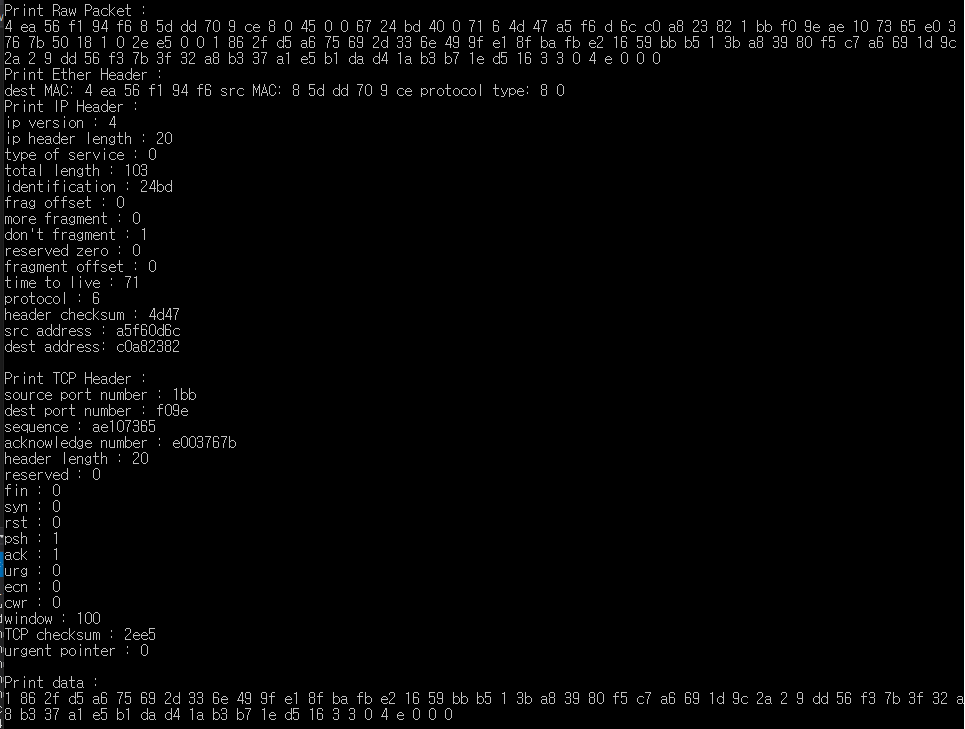


print\_data 함수에서는 데이터만 출력하기 때문에 헤더들의 총 길이가 저장된 tlength부터 pkt\_data의 끝인 caplen까지 pkt\_data의 값을 출력하도록 만들어줬다.





그대로 출력을 하면 너무 많은 패킷이 출력되기 때문에 count변수를 만들어서 패킷 20개만 출력하도록 만들어줬다.



프로그램을 실행하고 크롬에서 [www.inha.ac.kr](http://www.inha.ac.kr)에 들어가본 결과, 모든 정보가 정상적으로 출력됐다.