Libpcap 사용하기

노광민

dalgu2 (at) kldp.org

libpcap의 정의와 사용법, 응용 등을 제시한다.

고친 과정

고침 0.42002-01-01고친이 Kwang-Min Noh

Text 문서를 DocBook으로 수정 및 응용 부분 추가

1. Libpcap(Portable Packet Capturing Library)

패킷을 캡쳐하기 위한 도구로는 BPF(Berkeley Packet Filter), DLPI, NIT, SNOOP, SNIT, SOCK_PACKET, LSF(Linux Socket Filter), drain등 각 운영체제별로 다양한 도구가 있다. 하지만 여러분이 운영체제별로 패킷을 캡쳐하기 위한 위한 코드를 별도로 구성해야 한다면 시작하기도 전에 포기할지도 모른다. 하지만 이 모든 도구들을 수용하는 Portable한 API가 있는데 이것이 바로 libpcap (http://www.tcpdump.org)이다.

라이브러리 사용자는 운영체제의 각기 다른 datalink로의 접근 방법에 상관없이 libpcap을 이용하여 소기의 목적을 달성할 수 있다. libpcap을 이용한 대표적인 툴이 바로 tcpdump (http://www.tcpdump.org)이다.

2. 패킷을 캡쳐하기 위한 libpcap의 fuction

```
int main(int argc, char *argv[])
{
................
................

        if (device == NULL ) {
            if ( (device = pcap_lookupdev(ebuf) ) == NULL) {
                perror(ebuf);
                exit(-1);
            }
        }
```

```
        pd = pcap_open_live(device, snaplen, PROMISCUOUS, 1000, ebuf);
        if(pd == NULL) {
            perror(ebuf);
            exit(-1);
        }

        i = pcap_snapshot(pd);
        if(snaplen < i) {
            perror(ebuf);
            exit(-1);
        }

        if(pcap_lookupnet(device, &localnet, &netmask, ebuf) < 0) {
            perror(ebuf);
            exit(-1);
        }

        setuid(getuid());

        if(pcap_compile(pd, &fcode, filter_rule, 0, netmask) < 0) {
            perror(ebuf);
            exit(-1);
        }

        if(pcap_setfilter(pd, &fcode) < 0) {
            perror(ebuf);
            exit(-1);
        }

        fflush(stderr);

        printer = lookup_printer(pcap_datalink(pd));
        pcap_userdata = 0;

        if(pcap_loop(pd, packetcnt, printer, pcap_userdata) < 0) {
            perror("pcap_loop error");
            exit(-1);
        }

        pcap_close(pd);
exit(0);
}
```

2.1. pcap_lookupdev()

device = pcap_lookupdev(ebuf);

네트웍 디바이스를 가져오는 함수입니다. 패킷을 잡으려면 네트웍 디바이스를 지정해야 겠죠? 가능한 다비이스중 가장 번호가 낮은 디바이스를 가져오게 됩니다. 리눅스라면 eth0이겠죠... 다른 디바이스를 통해 패킷을 캡쳐하려면 이 함수를 사용하지 않고 프로그램의 입력으로 디바이스명을 가져오면 됩니다.

2.2. pcap_open_live()

pd = pcap_open_live(device, snaplen, PROMISCUOUS, 1000, ebuf);

위 함수는 실제 기기를 열어주는 기능을 하는 것으로 snaplen는 패킷당 저장할 바이스 수, 실제 datalink계층부터 패킷의 크기를 계산하여 원하는 부분만을 얻어오면 되는 것입니다. 헤더 정보만을 보고싶은데 쓸데없이 데이타까지 받을 필요는 없겠죠. 데이터까지 보고싶으면 snaplen를 크게 하면 됩니다. PROMISCUOUS는 1이며 네트웍 디바이스에 오는 모든 패킷을 받겠다는 의미입니다. 이 모드를 자세하게 설명하면 Ethernet은 모든 패킷이 broadcasting되며 일단 모든 네트웍 디바이스는 동일 네트웍내의 다른 호스트의 패킷도 일단 접하게 됩니다. 그러나, 네트웍 디바이스는 기본적으로 자신의 패킷만을 받게끔 되어있습니다. 그러므로 다른 호스트의 패킷은 버리게 되는 것입니다. 그러나 promiscuous모드로 디바이스 모드를 바꾸게 되면 모든 패킷을 받아들이게 되는 것입니다. 모든 네트워크 모니터링 프로그램들은 모두 이 모드를 사용하게 됩니다. 세 번째 인자는 패킷이 버퍼로 전달될 때 바로 전달되는 것이 아니라 위에서 명시한 시간을 넘겼을 때나 버퍼가 다 채워졌을 때 응용프로그램으로 전달되는 것입니다.

2.3. pcap_lookupnet()

pcap_lookupnet(device, &localnet, &netmask, ebuf)

열려진 패킷 캡쳐 디바이스에 네트웍 주소와 서브넷 마스크를 넘겨줍니다.

2.4. pcap_compile()

pcap_compile(pd, &fcode, filter_rule, 0, netmask)

정해진 필터룰에 의해 필터 프로그램을 컴파일하게 되는데 우리가 원하는 패킷은 필터룰을 주어야만 원하는 패킷만을 얻을 수 있습니다. 실제 tcpdump에서 사용하는 필터룰이 여기에서 쓰입니다. 예를 들면 "tcp port 80" 입니다.

자세한 필터룰에 대한 설명은 tcpdump의 메뉴얼을 보면 알 수 있습니다.

2.5. pcap_setfilter()

pcap_setfilter(pd, &fcode)

위는 앞서 컴파일한 필터 프로그램을 지정해 주는데 사용됩니다. 이렇게 하여 원하는 패킷을 얻을 준비를 하게 됩니다.

2.6. pcap_datalink()

printer = lookup_printer(pcap_datalink(pd));

위는 패킷 캡쳐 디바이스의 datalink계층의 종류를 넘겨 받아 이에 따른 적절한 함수포인터를 할당하게 됩니다.

2.7. pcap_loop()

pcap_loop(pd, packetcnt, printer, pcap_userdata)

실제 패킷을 잡아서 실행할 함수를 지정해 주는 함수입니다. packetcnt의 수만큼 패킷을 잡아서 잡을 때 마다 해당 패킷을 printer가 포인터하는 함수에게 전달하고 함수를 수행하게 됩니다.

packetcnt를 0으로 지정하면 무한대로 함수를 실행합니다.

2.8. pcap_next()

함수형 포인터를 사용하지 않고 함수를 호출할 때마다 패킷을 리턴해주는 함수로 앞의 pcap_datalink()와 pcap_loop()를 대체할 수 있습니다.

사용예는 응용부분4절에서 참고하십시오.

3. 사용 예

실제 libpcap을 이용하는 많은 응용이 있습니다. 위에서 언급한 tcpdump가 그 대표적인 예이며, 이것 외에 많은 네트워크 모니터링 툴, 공개 네트워크 IDS(Intrusion Detection System) 인 Snort, 패킷을 캡쳐하는 많은 응용들이 libpcap을 이용하여 다양한 OS에 포팅되어 있습니다. www.tcpdump.org/related.html (http://www.tcpdump.org/related.html)에 가면 응용의 예들을 볼 수 있습니다.

그럼 libpcap을 이용한 간단한 예를 보이겠습니다. 아래 소스는 ip기반 tcp, udp, icmp의 패킷을 잡아서 각 프로토콜 필드별로 프린트해주는 소스입니다.

먼저 IP, TCP, UDP의 각 HEADER FORMAT을 참조하시기 바랍니다.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Internet Datagram Header

<RFC791>

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP Header Format

<RFC793>

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
```

```
                 |                 |                 |
                 |     Length      |    Checksum     |
                 +--------+--------+--------+--------+
                 |
                 |          data octets ...
                 +---------------- ...

                      User Datagram Header Format
<RFC768>
```

```
#include <sys/time.h>
#include <netinet/in.h>
#include <net/ethernet.h>
#include <pcap/pcap.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/ip_icmp.h>

#define  PROMISCUOUS 1

struct   iphdr    *iph;
struct   tcphdr   *tcph;
struct   udphdr   *udph;
struct   icmp     *icmph;
static   pcap_t   *pd;
int sockfd;
int pflag;
int rflag;
int eflag;
int cflag;
int chcnt;

char    *device, *filter_rule;

void packet_analysis(unsigned char *, const struct pcap_pkthdr *,
                     const unsigned char *);

struct printer {
   pcap_handler f;
   int type;
```

```
};

/* datalink type에 따른 불리어질 함수들의
   목록들을 갖는 구조체
 Data-link level type codes.
#define DLT_NULL              0        no link-layer encapsulation
#define DLT_EN10MB       1        Ethernet (10Mb)
#define DLT_EN3MB             2        Experimental Ethernet (3Mb)
#define DLT_AX25              3        Amateur Radio AX.25
#define DLT_PRONET       4        Proteon ProNET Token Ring
#define DLT_CHAOS             5        Chaos
#define DLT_IEEE802       6        IEEE 802 Networks
#define DLT_ARCNET       7        ARCNET
#define DLT_SLIP              8        Serial Line IP
#define DLT_PPP          9        Point-to-point Protocol
#define DLT_FDDI              10       FDDI
#define DLT_ATM_RFC1483  11       LLC/SNAP encapsulated atm
#define DLT_RAW          12       raw IP
#define DLT_SLIP_BSDOS   13       BSD/OS Serial Line IP
#define DLT_PPP_BSDOS    14       BSD/OS Point-to-point Protocol
bpf.h 라는 헤더화일에 위와 같은 내용으로 정의되어 있다.         */

static struct printer printers[] = {
   { packet_analysis, DLT_IEEE802 },
   { packet_analysis, DLT_EN10MB  },
   { NULL, 0 },
};

/*  datalink type에 따라 수행될 함수를 결정하게 된다.
    이는 pcap_handler라는 함수형 포인터의 값으로 대입된다. */
static pcap_handler lookup_printer(int type)
{
struct printer *p;

for(p=printers; p->f; ++p)
if(type == p->type)
return p->f;

perror("unknown data link type");
}

/* pcap_loop()에 의해 패킷을 잡을 때마다 불려지는 함수
   pcap_handler가 이 함수를 포인터하고 있기 때문이다 */
void packet_analysis(unsigned char *user, const struct pcap_pkthdr *h,
                    const unsigned char *p)
{
int j, temp;
unsigned int length = h->len;
struct ether_header *ep;
```

```
unsigned short ether_type;
unsigned char *tcpdata, *udpdata,*icmpdata, temp_char;
register unsigned int i;

chcnt = 0;

if(rflag) {
while(length--) {
printf("%02x ", *(p++));
if( (++chcnt % 16) == 0 ) printf("\n");
}
fprintf(stdout, "\n");
return;
}

length -= sizeof(struct ether_header);

// ethernet header mapping
ep = (struct ether_header *)p;
// ethernet header 14 bytes를 건너 뛴 포인터
p += sizeof(struct ether_header);
// datalink type
ether_type = ntohs(ep->ether_type);

printf("\n");
// Ethernet frame이 IEEE802인경우 ether_type필드가 길이필드가 된다.
if(ether_type <= 1500) {
;
/*while(length--) {
if(++is_llchdr <= 3) {
fprintf(stdout,"%02x",*p++);
continue;
}
if(++next_line == 16) {
next_line = 0;
printf("\n");
}
printf("%02x",*p++);
}*/
}
else
{
if(eflag) {
printf("\n\n=================== Datalink layer ===================\n");
for(j=0; j<ETH_ALEN; j++) {
printf("%X", ep->ether_shost[j]);
if(j != 5) printf(":");
}
printf("  ------> ");
```

```
for(j=0; j<ETH_ALEN; j++){
printf("%X", ep->ether_dhost[j]);
if(j != 5) printf(":");
}
printf("\nether_type -> %x\n", ntohs(ep->ether_type));
}

iph = (struct iphdr *) p;
i = 0;
if (ntohs(ep->ether_type) == ETHERTYPE_IP) {        // ip 패킷인가?
printf("\n\n===================    IP HEADER   ===================\n");
printf("%s -----> ",   inet_ntoa(iph->saddr));
printf("%s\n", inet_ntoa(iph->daddr));
printf("Version:         %d\n", iph->version);
printf("Herder Length:   %d\n", iph->ihl);
printf("Service:         %#x\n",iph->tos);
printf("Total Length:    %d\n", ntohs(iph->tot_len));
printf("Identification : %d\n", ntohs(iph->id));
printf("Fragment Offset: %d\n", ntohs(iph->frag_off));
printf("Time to Live:    %d\n", iph->ttl);
printf("Checksum:        %d\n", ntohs(iph->check));

if(iph->protocol == IPPROTO_TCP) {
tcph = (struct tcphdr *) (p + iph->ihl * 4);
// tcp data는
tcpdata = (unsigned char *) (p + (iph->ihl*4) + (tcph->doff * 4));
printf("\n\n===================   TCP HEADER   ===================\n");
printf("Source Port:              %d\n", ntohs(tcph->source));
printf("Destination Port:         %d\n", ntohs(tcph->dest));
printf("Sequence Number:          %d\n", ntohl(tcph->seq));
printf("Acknowledgement Number:   %d\n", ntohl(tcph->ack_seq));
printf("Data Offset:              %d\n", tcph->doff);
printf("Window:                   %d\n", ntohs(tcph->window));
printf("URG:%d ACK:%d PSH:%d RST:%d SYN:%d FIN:%d\n",
tcph->urg, tcph->ack, tcph->psh, tcph->rst,
tcph->syn, tcph->fin, ntohs(tcph->check),
ntohs(tcph->urg_ptr));
printf("\n===================   TCP DATA(HEX)  =================\n");
chcnt = 0;
for(temp = (iph->ihl * 4) + (tcph->doff * 4); temp <= ntohs(iph-
>tot_len) - 1; temp++) {
printf("%02x ", *(tcpdata++));
if( (++chcnt % 16) == 0 ) printf("\n");
}
if (pflag) {
   tcpdata = (unsigned char *) (p + (iph->ihl*4) + (tcph->doff * 4));
   printf("\n===================   TCP DATA(CHAR)  =================\n");
   for(temp = (iph->ihl * 4) + (tcph->doff * 4); temp <= ntohs(iph-
>tot_len) - 1; temp++) {
```

```
temp_char = *tcpdata;
if ( (temp_char == 0x0d) && ( *(tcpdata+1) == 0x0a ) ) {
fprintf(stdout,"\n");
tcpdata += 2;
temp++;
continue;
}
temp_char = ( ( temp_char >= ' ' ) && ( temp_char < 0x7f ) )? temp_char : '.';
printf("%c", temp_char);
tcpdata++;
   }
}
printf("\n>>>>> End of Data >>>>>\n");
}
else if(iph->protocol == IPPROTO_UDP) {
udph = (struct udphdr *) (p + iph->ihl * 4);
udpdata = (unsigned char *) (p + iph->ihl*4) + 8;
printf("\n==================== UDP HEADER =====================\n");
printf("Source Port :      %d\n",ntohs(udph->source));
printf("Destination Port : %d\n", ntohs(udph->dest));
printf("Length :           %d\n", ntohs(udph->len));
printf("Checksum :         %x\n", ntohs(udph->check));
printf("\n===================  UDP DATA(HEX)  ================\n");
chcnt = 0;
for(temp = (iph->ihl*4)+8; temp<=ntohs(iph->tot_len) -1; temp++) {
   printf("%02x ", *(udpdata++));
   if( (++chcnt % 16) == 0) printf("\n");
}

udpdata = (unsigned char *) (p + iph->ihl*4) + 8;
if(pflag) {
printf("\n===================  UDP DATA(CHAR)  ================\n");
for(temp = (iph->ihl*4)+8; temp<=ntohs(iph->tot_len) -1; temp++)  {
temp_char = *udpdata;
if ( (temp_char == 0x0d) && ( *(udpdata+1) == 0x0a ) ) {
fprintf(stdout,"\n");
udpdata += 2;
temp++;
continue;
}
temp_char = ( ( temp_char >= ' ' ) && ( temp_char < 0x7f ) )? temp_char : '.';
printf("%c", temp_char);
udpdata++;
}
}

printf("\n>>>>> End of Data >>>>>\n");
}
else if(iph->protocol == IPPROTO_ICMP) {
```

```
icmph = (struct icmp *) (p + iph->ihl * 4);
icmpdata = (unsigned char *) (p + iph->ihl*4) + 8;
printf("\n\n===================   ICMP HEADER   ===================\n");
printf("Type :                     %d\n", icmph->icmp_type);
printf("Code :                     %d\n", icmph->icmp_code);
printf("Checksum :                 %02x\n", icmph->icmp_cksum);
printf("ID :                       %d\n", icmph->icmp_id);
printf("Seq :                      %d\n", icmph->icmp_seq);
printf("\n===================   ICMP DATA(HEX)  =================\n");
chcnt = 0;
for(temp = (iph->ihl * 4) + 8; temp <= ntohs(iph->tot_len) - 1; temp++) {
printf("%02x ", *(icmpdata++));
if( (++chcnt % 16) == 0 ) printf("\n");
}
printf("\n>>>>> End of Data >>>>>\n");
   }
}
}
}

void sig_int(int sig)
{
    printf("Bye!!\n");
    pcap_close(pd);
    close(sockfd);
    exit(0);
}

void usage(void)
{
    fprintf(stdout," Usage : noh_pa filter_rule [-pch]\n");
    fprintf(stdout,"         -p  :  데이타를 문자로 출력한다.\n");
    fprintf(stdout,"         -c  :  주어진 숫자만큼의 패킷만 덤프한다\n");
    fprintf(stdout,"         -e  :  datalink layer를 출력한다.\n");
    fprintf(stdout,"         -r  :  잡은 패킷을 생으로 찍는다.\n");
    fprintf(stdout,"         -h  :  사용법\n");
}

int main(int argc, char *argv[])
{
struct  bpf_program fcode;
pcap_handler printer;
char    ebuf[PCAP_ERRBUF_SIZE];
int     c, i, snaplen = 1514, size, packetcnt;
bpf_u_int32 myself, localnet, netmask;
unsigned char   *pcap_userdata;

filter_rule = argv[1];          /* exam-
ple : "src host xxx.xxx.xxx.xxx and tcp port 80" */
```

```
signal(SIGINT,sig_int);

opterr = 0;

if(argc-1 < 1) {
usage();
exit(1);
}

while( (c = getopt(argc, argv,"i:c:pher")) != -1) {
switch(c) {
case 'i'  :
device = optarg;
break;
case 'p' :
pflag = 1;
break;
case 'c' :
cflag = 1;
packetcnt = atoi(optarg);
if(packetcnt <= 0) {
fprintf(stderr,"invalid number %s",optarg);
exit(1);
}
break;
case 'e' :
eflag = 1;
break;
case 'r' :
rflag = 1;
break;
case 'h' :
usage();
exit(1);
}
}

if (device == NULL ) {
if ( (device = pcap_lookupdev(ebuf) ) == NULL) {
perror(ebuf);
exit(-1);
}
}
fprintf(stdout, "device = %s\n", device);

pd = pcap_open_live(device, snaplen, PROMISCUOUS, 1000, ebuf);
if(pd == NULL) {
perror(ebuf);
```

```
exit(-1);
}

i = pcap_snapshot(pd);
if(snaplen < i) {
perror(ebuf);
exit(-1);
}

if(pcap_lookupnet(device, &localnet, &netmask, ebuf) < 0) {
perror(ebuf);
exit(-1);
}

setuid(getuid());

if(pcap_compile(pd, &fcode, filter_rule, 0, netmask) < 0) {
perror(ebuf);
exit(-1);
}

if(pcap_setfilter(pd, &fcode) < 0) {
perror(ebuf);
exit(-1);
}

fflush(stderr);

printer = lookup_printer(pcap_datalink(pd));
pcap_userdata = 0;

if(pcap_loop(pd, packetcnt, printer, pcap_userdata) < 0) {
perror("pcap_loop error");
exit(-1);
}

pcap_close(pd);
exit(0);
}
```

4. 응용 : icmp_toy

libpcap을 이용한 간단한 응용의 예를 보이도록 하겠습니다.

icmp_toy는 특정 호스트의 ip address를 입력으로 받아 그 호스트의 모든 접속을 차단해 버리는 프로그램으로 해당 호스트의 tcp, syn flag가 set된 패킷을 libpcap을 이용하여 캡쳐한 후 해당 패킷에 대한 icmp protocol unreachable 패킷을 해당 호스트에게 재전송하게 됩니다.

입력한 호스트는 icmp_toy가 구동중인 호스트와 같은 네트워크에 구성되어 있고, Dummy Hub 환경이여야만 제대로 작동하게 됩니다.

icmp packet은 아래와 같은 구조로 만들어주게 됩니다.

```
Destination Unreachable Message

    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Type      |     Code      |          Checksum             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                             unused                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      Internet Header + 64 bits of Original Data Datagram      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

    Type : 3
    Code : 2 = protocol unreachable
    <RFC792>
```

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/ip_icmp.h>
#include <net/ethernet.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/time.h>
#include <pcap.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

#define  PROMISCUOUS 1

static int nchild = 5;
char target_ip[16];

static pid_t *pids;
static pcap_t *pd;
```

```
unsigned short in_cksum (unsigned short *addr, int len)
{
  int nleft = len;
  int sum = 0;
  unsigned short *w = addr;
  unsigned short answer = 0;

  while (nleft > 1) {
      sum += *w++;
      nleft -= 2;
  }

  if (nleft == 1) {
      *(unsigned char *) (&answer) = *(unsigned char *) w;
      sum += answer;
  }

  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  answer = ~sum;
  return (answer);
}

void send_icmp (int sockfd, struct iphdr *iph, struct tcphdr *tcph)
{
  char buff[76];
  char data[28];

  int len;
  struct sockaddr send;
  struct icmp *icmp;
  struct sockaddr_in *willsend;

  willsend = (struct sockaddr_in *) &send;
  willsend->sin_family = AF_INET;
  willsend->sin_addr.s_addr = iph->saddr;

  fprintf (stdout, "A player\'s number is (%d) : Shot!! ---<-@ %s \n",
   getpid (), target_ip);

  icmp = (struct icmp *) buff;
  icmp->icmp_type = ICMP_DEST_UNREACH;
  icmp->icmp_code = ICMP_PROT_UNREACH;
  icmp->icmp_id = 0;
  icmp->icmp_seq = 0;

  memcpy (data, iph, 20);
  memcpy (data + 20, tcph, 8);
  memcpy (icmp->icmp_data, data, 28);
```

```
  len = 8 + 20 + 8;

  icmp->icmp_cksum = 0;
  icmp->icmp_cksum = in_cksum ((u_short *) icmp, len);
  sendto (sockfd, buff, len, 0, &send, sizeof (send));
}

void checkip (struct iphdr *iph, struct tcphdr *tcph)
{
  int sockfd;
  char source_ip[16];

  struct in_addr in;
  in.s_addr = iph->saddr;

  strncpy (source_ip, inet_ntoa (in), sizeof (source_ip));
  sockfd = socket (AF_INET, SOCK_RAW, IPPROTO_ICMP);

  if (strncmp (target_ip, source_ip, sizeof (source_ip)) == 0)
    send_icmp (sockfd, iph, tcph);

  close (sockfd);
}

void packet_info (char *user, int len)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  iph = (struct iphdr *) user;
  tcph = (struct tcphdr *) (user + iph->ihl * 4);
  checkip (iph, tcph);
}

void sig_int (int sig)
{
  int i;

  for (i = 0; i < nchild; i++)
    kill (pids[i], SIGTERM);

  while (wait (NULL) > 0)
    ;

  fprintf (stdout, "Bye !!\n");

  pcap_close(pd);
  exit (0);
```

```
}

pid_t child_make (int i, pcap_t * pd, int datalink)
{
  pid_t pid;
  void child_main (int, pcap_t *, int);

  if ((pid = fork ()) > 0) {
      return (pid);
  }

  child_main (i, pd, datalink);
  return 0; // ADD
}

void child_main (int i, pcap_t * pd, int datalink)
{
  void packet_loop (pcap_t *, int);

  printf ("CHILD %ld starting\n", (long) getpid ());
  packet_loop (pd, datalink);
}

char *next_pcap (pcap_t * pd, int *len)
{
  char *ptr;
  struct pcap_pkthdr hdr;

  while ((ptr = (char *) pcap_next (pd, &hdr)) == NULL);

  *len = hdr.caplen;
  return (ptr);
}

void packet_loop (pcap_t * pd, int datalink)
{
  int len;
  char *ptr;

  for (;;) {
      ptr = next_pcap (pd, &len);
      switch (datalink) {
case DLT_EN10MB:
   packet_info (ptr + 14, len - 14);
   break;
  }
  }
}
```

```
void usage (void)
{
  fprintf (stdout, "SYNOPSIS : icmp_toy xxx.xxx.xxx.xxx(target ip ad-
dress) \n");
}

int main (int argc, char *argv[])
{
   struct bpf_program fcode;
   char *device, *filter_rule;
   char ebuf[PCAP_ERRBUF_SIZE];
   int i, j, snaplen = 68;
   bpf_u_int32 localnet, netmask;

   signal (SIGINT, sig_int);

   if (argc < 2) {
   usage ();
   exit (1);
   }

   strncpy (target_ip, argv[1], sizeof (target_ip));
   filter_rule = "tcp and tcp[13:1] & 2 != 0";
   device = pcap_lookupdev (ebuf);
   if (device == NULL)
   {
   perror (ebuf);
   exit (1);
   }


   pd = pcap_open_live (device, snaplen, PROMISCUOUS, 1000, ebuf);

   if (pd == NULL)
   {
   perror (ebuf);
   exit (1);
   }

   i = pcap_snapshot (pd);

   if (snaplen < 1)
   {
   perror (ebuf);
   exit (1);
   }

   if (pcap_lookupnet (device, &localnet, &netmask, ebuf) < 0)
   {
```

```
    perror (ebuf);
    exit (1);
    }

    if (pcap_setfilter (pd, &fcode) < 0)
    {
    perror (ebuf);
    exit (1);
    }

    fflush (stderr);

    pids = calloc (nchild, sizeof (pid_t));
    for (j = 0; j < nchild; j++)
    pids[j] = child_make (j, pd, pcap_datalink (pd));

    for (;;)
    pause ();
}
```

5. 관련 링크

- Libpcap : ftp://ftp.ee.lbl.gov
- WinPcap (http://netgroup-serv.polito.it/winpcap/) : the Free Packet Capture Architecture for Windows
- Tcpdump (http://www.tcpdump.org) : libpcap을 이용한 대표적인 네트워크 모니터링 툴
- Raw IP Networking FAQ (http://kldp.org/Translations/Raw_IP_FAQ-KLDP) : Raw socket, packet capturing 등에 대한 빈번한 질문과 답
- 동작그만(DJSTOP) (http://djstop.kldp.org) : libpcap, libnet 등을 이용한 인터넷 접속제어 프로그램

6. 저작권 정보 및 기타

6.1. 책임의 한계

본 저자는 문서의 내용이 야기할 수 있는 어떠한 결과에 대해서도 책임을 지지 않습니다. 본 문서에서 내포하고 있는 정보들 및 예제들은 여러분이 알아서 활용하십시오. 비록 최선을 다 했으나 이 문서는 틀린 점이나 오류가 있을 수도 있습니다. 만약 여러분이 틀린 점을 발견했다면 꼭 저에게 알려 주시기 바랍니다.

6.2. 피드백

이 문서에 대한 발전적인 제안이나 수정사항, 문제점 등에 대한 피드백은 언제든지 환영합니다. djstop (at) orgio.net로 메일을 보내 주십시오.