Create your own packet sniffer i

A simple implementation of a packet sniffer in C on linux p using the libpcap library. This packet sniffer currently sniffs IP , TCP , ICMP and UDP packets. It can be modified to any protocol as needed just by introducing the header information in it.

Certain filters can be used too like port number and specific host etc.
e.g.

Expression	Description
ip	Capture all IP packets.
tcp	Capture only TCP packets.
tcp port 80	Capture only TCP packets with a port equal to 80.
ip host 10.1.2.3	Capture all IP packets to or from host 10.1.2.3.

It is a little modified version of sniffer from tcpdump [http://www.tcpdump.org/] website.

Note : To run this code you require root permissions.

Here's the code:

? [#]

```
   /*sniffer.c*/
   //To compile : gcc -o sniffer sniffer.c -lpcap
   //To run : ./sniffer [interface-name]

   #include <pcap.h>
   #include <stdio.h>
   #include <string.h>
   #include <stdlib.h>
   #include <ctype.h>
   #include <errno.h>
   #include <sys/types.h>
   #include <sys/socket.h>
   #include <netinet/in.h>
   #include <arpa/inet.h>
1
2  /* default snap length (maximum bytes per packet to capture)
3  */
4  #define SNAP_LEN 1518
5
```

Simple XML Generator i…
Traversing and Printing …
Simple password base…
Simple XML Parser in C…
Complete File Input Out…
Simple multithreading …
DES Implementation in …
'ls' command impleme…
Implementation of Map …
Hexadecimal to Integer …
IPC Shared Memory Im…
IPC Message Queue I…
SCTP Server Client Imp…
Secure Server Client us…
Custom strncat function…
Custom strncpy functio…
Custom string compare…
Adapter Design Pattern …
UDP Server Client Impl…
Creating a Daemon Pro…
Pipe from parent to child

```
/* ethernet headers are always exactly 14 bytes [1] */
#define SIZE_ETHERNET 14

/* Ethernet addresses are 6 bytes */
#define ETHER_ADDR_LEN 6

/* Ethernet header */
struct sniff_ethernet {
        u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination
host address */
        u_char  ether_shost[ETHER_ADDR_LEN];    /* source host
address */
         u_short ether_type;                     /* IP? ARP?
RARP? etc */
};

/* IP header */
struct sniff_ip {
         u_char  ip_vhl;                 /* version << 4 |
header length >> 2 */
        u_char  ip_tos;                 /* type of service */
        u_short ip_len;                 /* total length */
        u_short ip_id;                  /* identification */
         u_short ip_off;                    /* fragment offset
field */
         #define IP_RF 0x8000            /* reserved fragment
flag */
        #define IP_DF 0x4000            /* dont fragment flag
*/
        #define IP_MF 0x2000            /* more fragments flag
*/
            #define  IP_OFFMASK  0x1fff           /*  mask  for
fragmenting bits */
        u_char  ip_ttl;                 /* time to live */
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
          struct  in_addr  ip_src,ip_dst;   /* source  and  dest
address */
};
#define IP_HL(ip)               (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)                (((ip)->ip_vhl) >> 4)

/* TCP header */
typedef u_int tcp_seq;

struct sniff_tcp {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
         tcp_seq th_ack;                    /* acknowledgement
number */
         u_char  th_offx2;                /* data offset, rsvd
```

Custom String Tokeniz...

Find if a substring exist...

Custom malloc function...

Basic File Operations i...

Function Pointers in C

Circular Queue using A...

Merge Sort Implementa...

Selection Sort Impleme...

```
58 */
59 #define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
60         u_char  th_flags;
61         #define TH_FIN  0x01
62         #define TH_SYN  0x02
63         #define TH_RST  0x04
64         #define TH_PUSH 0x08
65         #define TH_ACK  0x10
66         #define TH_URG  0x20
67         #define TH_ECE  0x40
68         #define TH_CWR  0x80
69                          #define   TH_FLAGS
70 (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
71         u_short th_win;                 /* window */
72         u_short th_sum;                 /* checksum */
73         u_short th_urp;                 /* urgent pointer */
74 };
75
76 void
77 got_packet(u_char  *args,  const  struct  pcap_pkthdr  *header,
78 const u_char *packet);
79
80 void
81 print_payload(const u_char *payload, int len);
82
83 void
84 print_hex_ascii_line(const  u_char  *payload,  int  len,  int
85 offset);
86
87
88
89 /*
90  * print data in rows of 16 bytes: offset   hex   ascii
91  *
92  * 00000   47 45 54 20 2f 20 48 54  54 50 2f 31 2e 31 0d 0a
93 GET / HTTP/1.1..
94  */
95 void
96 print_hex_ascii_line(const u_char *payload, int len, int offset)
97 {
98
99  int i;
100 int gap;
101 const u_char *ch;
102
103 /* offset */
104 printf("%05d   ", offset);
105
106 /* hex */
107 ch = payload;
108 for(i = 0; i < len; i++) {
109  printf("%02x ", *ch);
```

```
110  ch++;
111  /* print extra space after 8th byte for visual aid */
112  if (i == 7)
113   printf(" ");
114 }
115 /* print space to handle line less than 8 bytes */
116 if (len < 8)
117  printf(" ");
118
119 /* fill hex gap with spaces if not full line */
120 if (len < 16) {
121  gap = 16 - len;
122  for (i = 0; i < gap; i++) {
123   printf("   ");
124  }
125 }
126 printf("   ");
127
128 /* ascii (if printable) */
129 ch = payload;
130 for(i = 0; i < len; i++) {
131  if (isprint(*ch))
132   printf("%c", *ch);
133  else
134   printf(".");
135  ch++;
136 }
137
138 printf("\n");
139
140return;
141}
142
143/*
144 * print packet payload data (avoid printing binary data)
145 */
146void
147print_payload(const u_char *payload, int len)
148{
149
150 int len_rem = len;
151 int line_width = 16;   /* number of bytes per line */
152 int line_len;
153 int offset = 0;     /* zero-based offset counter */
154 const u_char *ch = payload;
155
156 if (len <= 0)
157  return;
158
159 /* data fits on one line */
160 if (len <= line_width) {
161  print_hex_ascii_line(ch, len, offset);
```

```
 return;
}

/* data spans multiple lines */
for ( ;; ) {
 /* compute current line length */
 line_len = line_width % len_rem;
 /* print line */
 print_hex_ascii_line(ch, line_len, offset);
 /* compute total remaining */
 len_rem = len_rem - line_len;
 /* shift pointer to remaining bytes to print */
 ch = ch + line_len;
 /* add offset */
 offset = offset + line_width;
 /* check if we have line width chars or less */
 if (len_rem <= line_width) {
  /* print last line and get out */
  print_hex_ascii_line(ch, len_rem, offset);
  break;
 }
}

return;
}

/*
 * dissect/print packet
 */
void
got_packet(u_char  *args,  const  struct  pcap_pkthdr  *header,
const u_char *packet)
{

 static int count = 1;                   /* packet counter */

 /* declare pointers to packet headers */
 const struct sniff_ethernet *ethernet;  /* The ethernet header
[1] */
 const struct sniff_ip *ip;              /* The IP header */
 const struct sniff_tcp *tcp;            /* The TCP header */
 const char *payload;                    /* Packet payload */

 int size_ip;
 int size_tcp;
 int size_payload;

 printf("\nPacket number %d:\n", count);
 count++;

 /* define ethernet header */
 ethernet = (struct sniff_ethernet*)(packet);
```

```
214
215 /* define/compute ip header offset */
216 ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
217 size_ip = IP_HL(ip)*4;
218 if (size_ip < 20) {
219    printf("     *  Invalid  IP  header  length:  %u  bytes\n",
220size_ip);
221  return;
222 }
223
224 /* print source and destination IP addresses */
225 printf("       From: %s\n", inet_ntoa(ip->ip_src));
226 printf("         To: %s\n", inet_ntoa(ip->ip_dst));
227
228 /* determine protocol */
229 switch(ip->ip_p) {
230  case IPPROTO_TCP:
231   printf("   Protocol: TCP\n");
232   break;
233  case IPPROTO_UDP:
234   printf("   Protocol: UDP\n");
235   return;
236  case IPPROTO_ICMP:
237   printf("   Protocol: ICMP\n");
238   return;
239  case IPPROTO_IP:
240   printf("   Protocol: IP\n");
241   return;
242  default:
243   printf("   Protocol: unknown\n");
244   return;
245 }
246
247 /*
248  *  OK, this packet is TCP.
249  */
250
251 /* define/compute tcp header offset */
252 tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
253 size_tcp = TH_OFF(tcp)*4;
254 if (size_tcp < 20) {
255    printf("     *  Invalid  TCP  header  length:  %u  bytes\n",
256size_tcp);
257  return;
258 }
259
260 printf("   Src port: %d\n", ntohs(tcp->th_sport));
261 printf("   Dst port: %d\n", ntohs(tcp->th_dport));
262
263 /* define/compute tcp payload (segment) offset */
264  payload  =  (u_char  *)(packet  +  SIZE_ETHERNET  +  size_ip  +
265size_tcp);
```

```
266
267 /* compute tcp payload (segment) size */
268 size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);
269
270 /*
271  * Print payload data; it might be binary, so don't just
272  * treat it as a string.
273  */
274 if (size_payload > 0) {
275  printf("   Payload (%d bytes):\n", size_payload);
276  print_payload(payload, size_payload);
277 }
278
279return;
280}
281
282int main(int argc, char **argv)
283{
284
285 char *dev = NULL;   /* capture device name */
286 char errbuf[PCAP_ERRBUF_SIZE];  /* error buffer */
287 pcap_t *handle;    /* packet capture handle */
288
289 char filter_exp[] = "ip";  /* filter expression */
290  struct  bpf_program  fp;       /*  compiled  filter  program
291(expression) */
292 bpf_u_int32 mask;   /* subnet mask */
293 bpf_u_int32 net;   /* ip */
294 int num_packets ;   /* number of packets to capture */
295
296 /* check for capture device name on command-line */
297 if (argc == 2) {
298  dev = argv[1];
299 }
300 else if (argc > 3) {
301       fprintf(stderr,   "error:   unrecognized   command-line
302options\n\n");
303 printf("Usage: %s [interface]\n", argv[0]);
304 printf("\n");
305 printf("Options:\n");
306  printf("        interface       Listen  on  <interface>  for
307packets.\n");
308 printf("\n");
309  exit(EXIT_FAILURE);
310 }
311 else {
312  /* find a capture device if not specified on command-line */
313  dev = pcap_lookupdev(errbuf);
314  if (dev == NULL) {
315   fprintf(stderr, "Couldn't find default device: %s\n",
316       errbuf);
317   exit(EXIT_FAILURE);
```

```
 }
}
printf("\nEnter no. of packets you want to capture: ");
       scanf("%d",&num_packets);
        printf("\nWhich kind of packets you want to capture :
");
       scanf("%s",filter_exp);
/* get network number and mask associated with capture device
*/
if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
 fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
     dev, errbuf);
 net = 0;
 mask = 0;
}

/* print capture info */
printf("Device: %s\n", dev);
printf("Number of packets: %d\n", num_packets);
printf("Filter expression: %s\n", filter_exp);

/* open capture device */
handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
if (handle == NULL) {
   fprintf(stderr,  "Couldn't  open  device  %s:  %s\n",  dev,
errbuf);
 exit(EXIT_FAILURE);
}

/* make sure we're capturing on an Ethernet device [2] */
if (pcap_datalink(handle) != DLT_EN10MB) {
 fprintf(stderr, "%s is not an Ethernet\n", dev);
 exit(EXIT_FAILURE);
}

/* compile the filter expression */
if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
 fprintf(stderr, "Couldn't parse filter %s: %s\n",
     filter_exp, pcap_geterr(handle));
 exit(EXIT_FAILURE);
}

/* apply the compiled filter */
if (pcap_setfilter(handle, &fp) == -1) {
 fprintf(stderr, "Couldn't install filter %s: %s\n",
     filter_exp, pcap_geterr(handle));
 exit(EXIT_FAILURE);
}

/* now we can set our callback function */
pcap_loop(handle, num_packets, got_packet, NULL);
```

```
 /* cleanup */
 pcap_freecode(&fp);
 pcap_close(handle);

 printf("\nCapture complete.\n");

return 0;
}
```

Posted 23rd October 2010 by Varun Gupta

Labels: Networking