

```
1  #include <pcap.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <ctype.h>
6  #include <errno.h>
7  #include <sys/types.h>
8  #include <sys/socket.h>
9  #include <netinet/in.h>
10 #include <arpa/inet.h>
11
12 /* default snap length (maximum bytes per packet to capture) */
13 #define SNAP_LEN 1518
14
15 /* ethernet headers are always exactly 14 bytes [1] */
16 #define SIZE_ETHERNET 14
17
18 /* Ethernet addresses are 6 bytes */
19 #define ETHER_ADDR_LEN 6
20
21 /* Ethernet header */
22 struct sniff_ethernet {
23     u_char ether_dhost[ETHER_ADDR_LEN]; /* destination host address */
24     u_char ether_shost[ETHER_ADDR_LEN]; /* source host address */
25     u_short ether_type; /* IP? ARP? RARP? etc */
26 };
27
28 /* IP header */
29 struct sniff_ip {
30     u_char ip_vhl; /* version << 4 | header length >> 2
31     */
32     u_char ip_tos; /* type of service */
33     u_short ip_len; /* total length */
34     u_short ip_id; /* identification */
35     u_short ip_off; /* fragment offset field */
36     #define IP_RF 0x8000 /* reserved fragment flag */
37     #define IP_DF 0x4000 /* dont fragment flag */
38     #define IP_MF 0x2000 /* more fragments flag */
39     #define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
40     u_char ip_ttl; /* time to live */
41     u_char ip_p; /* protocol */
42     u_short ip_sum; /* checksum */
43     struct in_addr ip_src, ip_dst; /* source and dest address */
44 };
45 #define IP_HL(ip) (((ip)->ip_vhl) & 0x0f)
46 #define IP_V(ip) (((ip)->ip_vhl) >> 4)
47
48 /* TCP header */
49 typedef u_int tcp_seq;
50
51 struct sniff_tcp {
52     u_short th_sport; /* source port */
53     u_short th_dport; /* destination port */
54     tcp_seq th_seq; /* sequence number */
55     tcp_seq th_ack; /* acknowledgement number */
56     u_char th_offx2; /* data offset, rsvd */
57     #define TH_OFF(th) (((th)->th_offx2 & 0xf0) >> 4)
58     u_char th_flags;
59     #define TH_FIN 0x01
60     #define TH_SYN 0x02
61     #define TH_RST 0x04
62     #define TH_PUSH 0x08
63     #define TH_ACK 0x10
64     #define TH_URG 0x20
65     #define TH_ECE 0x40
66     #define TH_CWR 0x80
```

```

66 #define TH_FLAGS      (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
67     u_short th_win;      /* window */
68     u_short th_sum;      /* checksum */
69     u_short th_urp;      /* urgent pointer */
70 };
71
72 void
73 got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char
    *packet);
74
75 void
76 print_payload(const u_char *payload, int len);
77
78 void
79 print_hex_ascii_line(const u_char *payload, int len, int offset);
80
81 void
82 print_app_banner(void);
83
84 void
85 print_app_usage(void);
86
87
88 /*
89  * print help text
90  */
91 void
92 print_app_usage(void)
93 {
94     printf("\n");
95     printf("Options:\n");
96     printf("    interface    Listen on <interface> for packets.\n");
97     printf("\n");
98
99     return;
100 }
101
102 /*
103  * print data in rows of 16 bytes: offset  hex  ascii
104  *
105  * 000000  47 45 54 20 2f 20 48 54  54 50 2f 31 2e 31 0d 0a  GET / HTTP/1.1..
106  */
107 void
108 print_hex_ascii_line(const u_char *payload, int len, int offset)
109 {
110
111     int i;
112     int gap;
113     const u_char *ch;
114
115     /* offset */
116     printf("%05d  ", offset);
117
118     /* hex */
119     ch = payload;
120     for (i = 0; i < len; i++) {
121         printf("%02x ", *ch);
122         ch++;
123         /* print extra space after 8th byte for visual aid */
124         if (i == 7)
125             printf(" ");
126     }
127     /* print space to handle line less than 8 bytes */
128     if (len < 8)
129         printf(" ");
130

```

```

131         /* fill hex gap with spaces if not full line */
132         if (len < 16) {
133             gap = 16 - len;
134             for (i = 0; i < gap; i++) {
135                 printf(" ");
136             }
137         }
138         printf(" ");
139
140         /* ascii (if printable) */
141         ch = payload;
142         for (i = 0; i < len; i++) {
143             if (isprint(*ch))
144                 printf("%c", *ch);
145             else
146                 printf(".");
147             ch++;
148         }
149         printf("\n");
150
151         return;
152     }
153 }
154
155 /*
156  * print packet payload data (avoid printing binary data)
157  */
158 void
159 print_payload(const u_char *payload, int len)
160 {
161     int len_rem = len;
162     int line_width = 16; /* number of bytes per line */
163     int line_len;
164     int offset = 0; /* zero-based offset
165 counter */
166     const u_char *ch = payload;
167
168     if (len <= 0)
169         return;
170
171     /* data fits on one line */
172     if (len <= line_width) {
173         print_hex_ascii_line(ch, len, offset);
174         return;
175     }
176
177     /* data spans multiple lines */
178     for ( ;; ) {
179         /* compute current line length */
180         line_len = line_width % len_rem;
181         /* print line */
182         print_hex_ascii_line(ch, line_len, offset);
183         /* compute total remaining */
184         len_rem = len_rem - line_len;
185         /* shift pointer to remaining bytes to print */
186         ch = ch + line_len;
187         /* add offset */
188         offset = offset + line_width;
189         /* check if we have line width chars or less */
190         if (len_rem <= line_width) {
191             /* print last line and get out */
192             print_hex_ascii_line(ch, len_rem, offset);
193             break;
194         }
195     }

```

```
196
197     return;
198 }
199
200 /*
201  * dissect/print packet
202  */
203 void
204 got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char
205 *packet)
206 {
207     static int count = 1;           /* packet counter */
208
209     /* declare pointers to packet headers */
210     const struct sniff_ethernet *ethernet; /* The ethernet header [1] */
211     const struct sniff_ip *ip;           /* The IP header */
212     const struct sniff_tcp *tcp;         /* The TCP header */
213     const char *payload;                 /* Packet payload */
214
215     int size_ip;
216     int size_tcp;
217     int size_payload;
218
219     printf("\nPacket number %d:\n", count);
220     count++;
221
222     /* define ethernet header */
223     ethernet = (struct sniff_ethernet*)(packet);
224
225     /* define/compute ip header offset */
226     ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
227     size_ip = IP_HL(ip) * 4;
228     if (size_ip < 20) {
229         printf("    * Invalid IP header length: %u bytes\n", size_ip);
230         return;
231     }
232
233     /* print source and destination IP addresses */
234     printf("        From: %s\n", inet_ntoa(ip->ip_src));
235     printf("        To: %s\n", inet_ntoa(ip->ip_dst));
236
237     /* determine protocol */
238     switch (ip->ip_p) {
239     case IPPROTO_TCP:
240         printf("    Protocol: TCP\n");
241         break;
242     case IPPROTO_UDP:
243         printf("    Protocol: UDP\n");
244         return;
245     case IPPROTO_ICMP:
246         printf("    Protocol: ICMP\n");
247         return;
248     case IPPROTO_IP:
249         printf("    Protocol: IP\n");
250         return;
251     default:
252         printf("    Protocol: unknown\n");
253         return;
254     }
255
256     /*
257      * OK, this packet is TCP.
258      */
259
260     /* define/compute tcp header offset */
```

```

261         tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
262         size_tcp = TH_OFF(tcp) * 4;
263         if (size_tcp < 20) {
264             printf(" * Invalid TCP header length: %u bytes\n",
size_tcp);
265             return;
266         }
267
268         printf(" Src port: %d\n", ntohs(tcp->th_sport));
269         printf(" Dst port: %d\n", ntohs(tcp->th_dport));
270         printf(" Checksum: %d\n", ntohs(tcp->th_sum));
271
272         /* define/compute tcp payload (segment) offset */
273         payload = (u_char*)(packet + SIZE_ETHERNET + size_ip + size_tcp);
274
275         /* compute tcp payload (segment) size */
276         size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);
277
278         /*
279          * Print payload data; it might be binary, so don't just
280          * treat it as a string.
281          */
282         if (size_payload > 0) {
283             printf(" Payload (%d bytes):\n", size_payload);
284             print_payload(payload, size_payload);
285         }
286
287         return;
288     }
289
290     int main(int argc, char **argv)
291     {
292
293         char *dev = NULL; /* capture device name */
294         char errbuf[PCAP_ERRBUF_SIZE]; /* error buffer */
295         pcap_t *handle; /* packet capture handle */
296
297         char filter_exp[] = "ip"; /* filter expression [3] */
298         struct bpf_program fp; /* compiled filter program
(expression) */
299         bpf_u_int32 mask; /* subnet mask */
300         bpf_u_int32 net; /* ip */
301         int num_packets = 10; /* number of packets to
capture */
302
303         /* check for capture device name on command-line */
304         if (argc == 2) {
305             dev = argv[1];
306         }
307         else if (argc > 2) {
308             fprintf(stderr, "error: unrecognized command-line
options\n\n");
309             print_app_usage();
310             exit(EXIT_FAILURE);
311         }
312         else {
313             /* find a capture device if not specified on command-line */
314             dev = pcap_lookupdev(errbuf);
315             if (dev == NULL) {
316                 fprintf(stderr, "Couldn't find default device: %s\n",
errbuf);
317                 exit(EXIT_FAILURE);
318             }
319         }
320     }
321
322     /* get network number and mask associated with capture device */

```

```
323     if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
324         fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
325             dev, errbuf);
326         net = 0;
327         mask = 0;
328     }
329
330     /* print capture info */
331     printf("Device: %s\n", dev);
332     printf("Number of packets: %d\n", num_packets);
333     printf("Filter expression: %s\n", filter_exp);
334
335     /* open capture device */
336     handle = pcap_open_offline("package.pcap", errbuf);
337     if (handle == NULL) {
338         fprintf(stderr, "Couldn't open file %s: %s\n", dev, errbuf);
339         exit(EXIT_FAILURE);
340     }
341
342     /* make sure we're capturing on an Ethernet device [2] */
343     if (pcap_datalink(handle) != DLT_EN10MB) {
344         fprintf(stderr, "%s is not an Ethernet\n", dev);
345         exit(EXIT_FAILURE);
346     }
347
348     /* compile the filter expression */
349     if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
350         fprintf(stderr, "Couldn't parse filter %s: %s\n",
351             filter_exp, pcap_geterr(handle));
352         exit(EXIT_FAILURE);
353     }
354
355     /* apply the compiled filter */
356     if (pcap_setfilter(handle, &fp) == -1) {
357         fprintf(stderr, "Couldn't install filter %s: %s\n",
358             filter_exp, pcap_geterr(handle));
359         exit(EXIT_FAILURE);
360     }
361
362     /* now we can set our callback function */
363     pcap_loop(handle, num_packets, got_packet, NULL);
364
365     /* cleanup */
366     pcap_freecode(&fp);
367     pcap_close(handle);
368
369     printf("\nCapture complete.\n");
370
371     return 0;
372 }
```