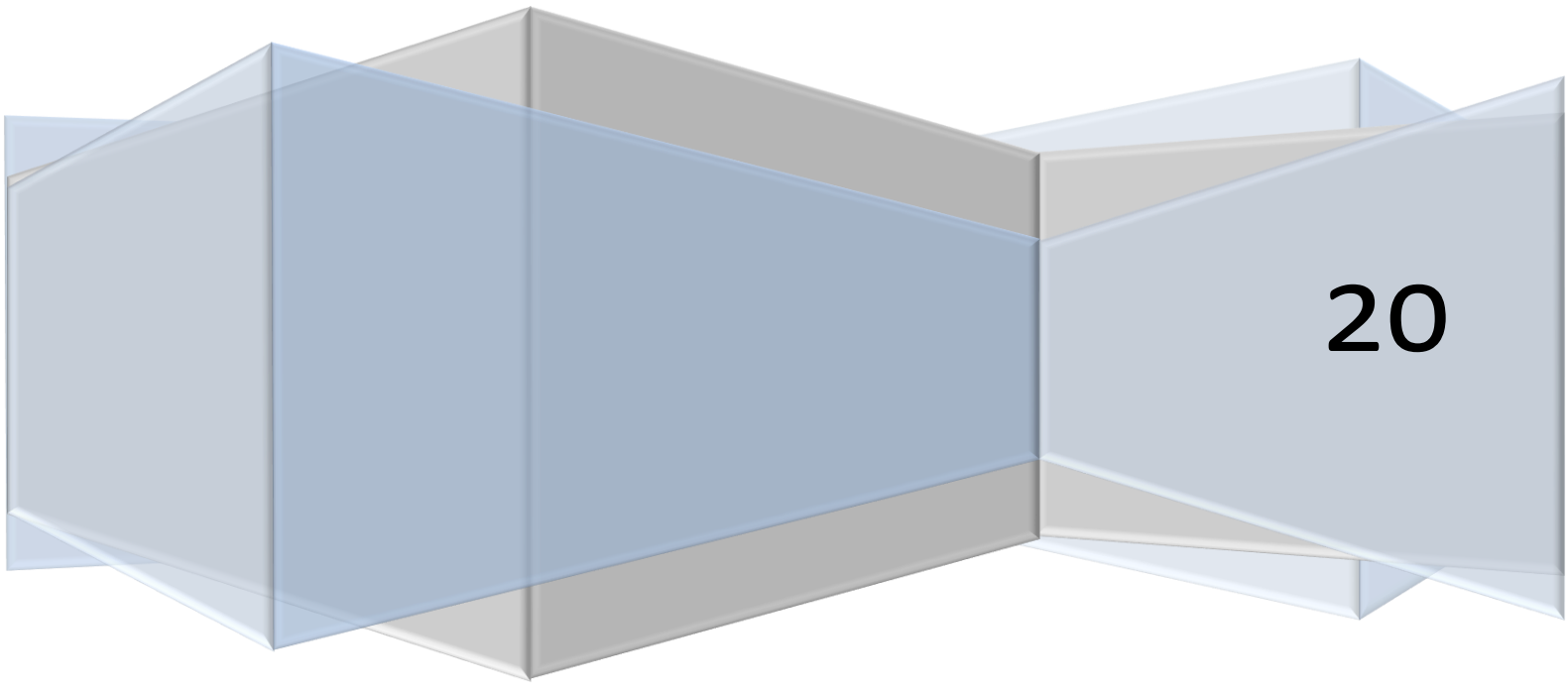
DEEP PACKET INSPECTION LAB

FTP Parser

DPI LAB

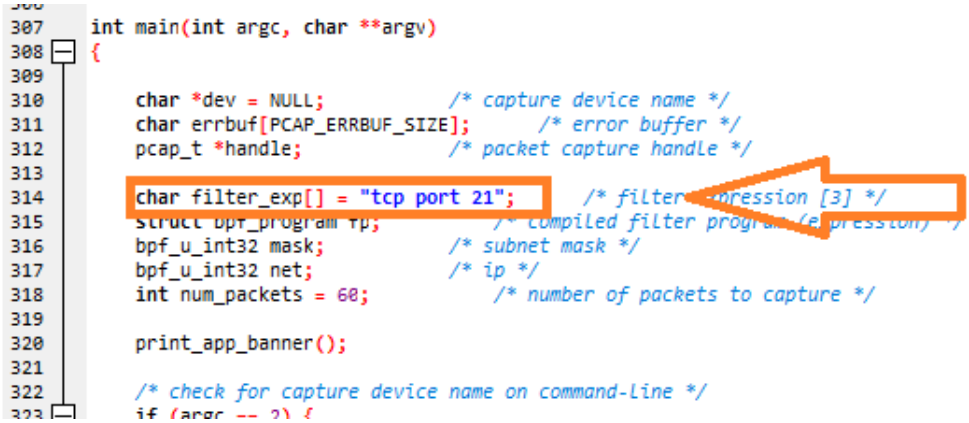
SUBMITTED BY: Shaheem Naqvi



FTP Parser

Extraction of FTP Protocol:

My first task was extraction of ftp protocol by using sniffex.c code which was described briefly in workshop of protocol extraction using C language given by Sir Siraj. For capturing ftp packets i simply used filter expression as “tcp port 21” and code started to check protocol id and port number to filter out ftp packets.



```
307  int main(int argc, char **argv)
308  {
309
310      char *dev = NULL;           /* capture device name */
311      char errbuf[PCAP_ERRBUF_SIZE];      /* error buffer */
312      pcap_t *handle;             /* packet capture handle */
313
314      char filter_exp[] = "tcp port 21";     /* filter expression [3] */
315      struct bpf_program fp;          /* compiled filter program (expression) */
316      bpf_u_int32 mask;           /* subnet mask */
317      bpf_u_int32 net;            /* ip */
318      int num_packets = 60;           /* number of packets to capture */
319
320      print_app_banner();
321
322      /* check for capture device name on command-line */
323      if (argc == 2) {
```

Man page of filter expression gives number of other formats can be used in this scope.

pcap-filter - packet filter syntax:

DESCRIPTION:

pcap_compile() is used to compile a string into a filter program. The resulting filter program can then be applied to some stream of packets to determine which packets will be supplied to **pcap_loop**(3PCAP), **pcap_dispatch**(3PCAP), **pcap_next**(3PCAP), or **pcap_next_ex**(3PCAP).

The *filter expression* consists of one or more *primitives*. Primitives usually consist of an *id* (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

type:

type qualifiers say what kind of thing the id name or number refers to. Possible types are **host**, **net**, **port** and **portrange**. E.g., `**host** foo', `**net** 128.3', `**port** 20', `**portrange** 6000-6008'. If there is no type qualifier, **host** is assumed.

dir:

dir qualifiers specify a particular transfer direction to and/or from *id*. Possible directions are **src**, **dst**, **src or dst**, **src and dst**, **ra**, **ta**, **addr1**, **addr2**, **addr3**, and **addr4**. E.g., `**src** foo', `**dst net** 128.3', `**src or dst port** ftp-data'. If there is no dir qualifier, `**src or dst**' is assumed. The **ra**, **ta**, **addr1**, **addr2**, **addr3**, and **addr4** qualifiers are only valid for IEEE 802.11 Wireless LAN link layers.

proto:

proto qualifiers restrict the match to a particular protocol. Possible protos are: **ether**, **fddi**, **tr**, **wlan**, **ip**, **ip6**, **arp**, **rarp**, **decnet**, **tcp** and **udp**. E.g., `**ether src** foo', `**arp net** 128.3', `**tcp port** 21', `**udp portrange** 7000-7009', `**wlan addr2** 0:2:3:4:5:6'. If there is no proto

qualifier, all protocols consistent with the type are assumed. E.g., `**src** foo' means `**(ip or arp or rarp) src** foo' (except the latter is not legal syntax), `**net** bar' means `**(ip or arp or rarp) net** bar' and `**port** 53' means `**(tcp or udp) port** 53'.

Output:

```
shaheem@shaheem-Naqvi:~$ gcc ftp21.c -o shah -lpcap
shaheem@shaheem-Naqvi:~$ sudo ./shah wlp0s20f3
sniffex - FTP Sniffer using libpcap
Copyright (c) 2020 DPI LAB
Deep Packet Inspection Lab

Device: wlp0s20f3
Number of packets: 100
Filter expression: tcp port 21


Packet number 1:
     From: 192.168.10.18
     To: 35.209.241.59
   Protocol: TCP
   Src port: 37968
   Dst port: 21


Packet number 2:
     From: 35.209.241.59
     To: 192.168.10.18
   Protocol: TCP
   Src port: 21
   Dst port: 37968
   L7 Protocol: FTP


Packet number 3:
     From: 192.168.10.18
     To: 35.209.241.59
   Protocol: TCP
   Src port: 37968
   Dst port: 21
```

Capturing of Client Packets:

To capture only client side commands during ftp session, I simply put a check on destination port which always will be 21 for ftp client.

```
267     tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
268     size_tcp = TH_OFF(tcp)*4;
269
270     if (size_tcp < 20) {
271         printf("   * Invalid TCP header length: %u bytes\n", size_tcp);
272         return;
273     }
274     /***capturing only destination port packets***/
275     if(ntohs(tcp->th_dport) == 21){
276
277     printf("   Src port: %d\n", ntohs(tcp->th_sport));
278     printf("   Dst port: %d\n", ntohs(tcp->th_dport));
279
280         if (ntohs(tcp->th_sport) == 21)
281         {   printf("   L7 Protocol: FTP\n");
282     }
283
284     if (ntohs(tcp->th_dport)== 21)
285     {   printf("   L7 Protocol: FTP\n");
286     }
287
```

OUTPUT:

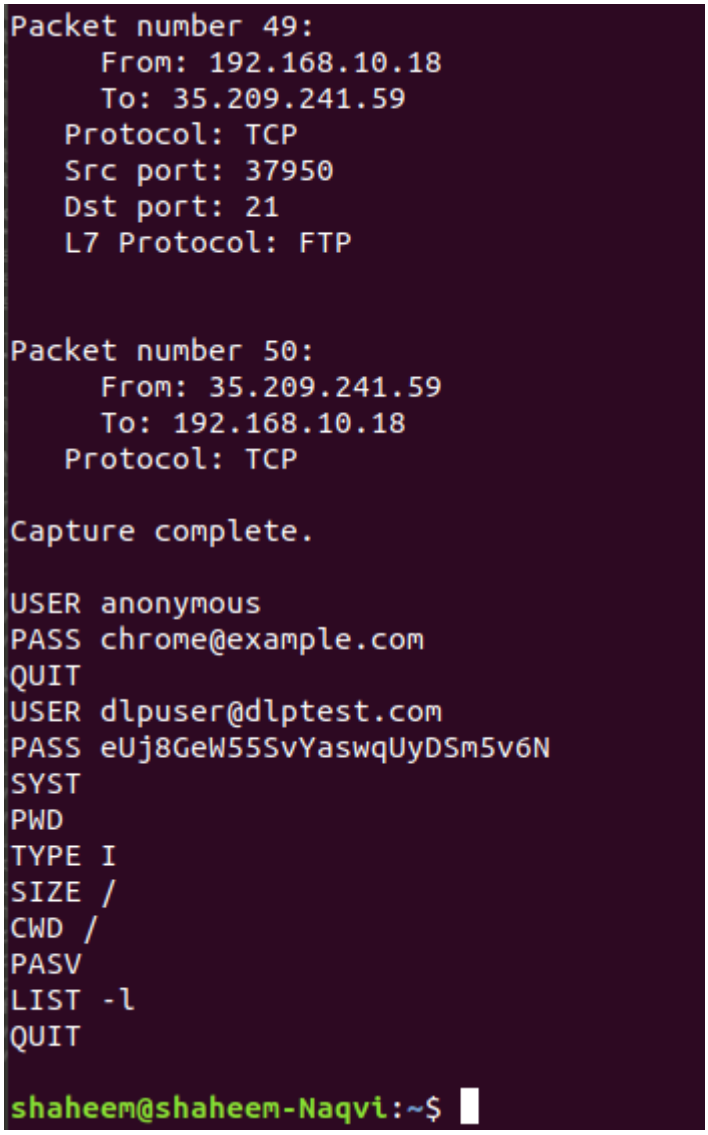
```
shaheem@shaheem-Naqvi:~$ gcc ftpclientcom.c -o shah -lpcap
shaheem@shaheem-Naqvi:~$ sudo ./shah wlp0s20f3
[sudo] password for shaheem:
sniffex - FTP Sniffer using libpcap
Copyright (c) 2020 DPI LAB
Deep Packet Inspection Lab

Device: wlp0s20f3
Number of packets: 50
Filter expression: tcp port 21


Packet number 1:
     From: 192.168.10.18
     To: 35.209.241.59
   Protocol: TCP
   Src port: 37948
   Dst port: 21
   L7 Protocol: FTP


Packet number 2:
     From: 192.168.10.18
     To: 35.209.241.59
   Protocol: TCP
   Src port: 37948
   Dst port: 21
   L7 Protocol: FTP
```

Creating Array of Client Side Commands:

To create an array which save each command of client sent to a server. I declared a global array and a variable "j" which count number of values added in array which help us to prevent over writing of an array.

```
138  void
139  print_hex_ascii_line(const u_char *payload, int len, int offset)
140  {
141      int i;
142      int gap;
143      const u_char *ch;
144      ch = payload;
145      for(i = 0; i < len; i++) {
146          if (isprint(*ch)){
147              printf("%c", *ch);
148
149          }
150          else
151              {printf(" ");}
152      s[j] =*ch;
153      ch++;
154      j++;
155      }
156  return;
157  }
158
```

OUTPUT:



Source Code:

```
1.  #define APP_NAME        "sniffex"
2.  #define APP_DESC        "FTP Sniffer using libpcap"
3.  #define APP_COPYRIGHT   "Copyright (c) 2020 DPI LAB"
4.  #define APP_DISCLAIMER  "Deep Packet Inspection Lab"
5.
6.  #include <pcap.h>
7.  #include <stdio.h>
8.  #include <string.h>
9.  #include <stdlib.h>
10. #include <ctype.h>
11. #include <errno.h>
12. #include <sys/types.h>
13. #include <sys/socket.h>
14. #include <netinet/in.h>
15. #include <arpa/inet.h>
16.
17. /* default snap length (maximum bytes per packet to capture) */
18. #define SNAP_LEN 1518
19.
```

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

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

#define IP_HL(ip)               (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)                (((ip)->ip_vhl) >> 4)

u_char s[300];
int j=0;
/* 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            /* don't 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 */
};


/* 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 */
    #define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)

        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};


```

```
85. void
86. got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet);
87.
88. void
89. print_payload(const u_char *payload, int len);
90.
91. void
92. print_hex_ascii_line(const u_char *payload, int len, int offset);
93.
94. void
95. print_app_banner(void);
96.
97. void
98. print_app_usage(void);
99.
100.        /*
101.         * app name/banner
102.         */
103.        void
104.        print_app_banner(void)
105.        {
106.
107.            printf("%s - %s\n", APP_NAME, APP_DESC);
108.            printf("%s\n", APP_COPYRIGHT);
109.            printf("%s\n", APP_DISCLAIMER);
110.            printf("\n");
111.
112.        return;
113.        }
114.
115.        /*
116.         * print help text
117.         */
118.        void
119.        print_app_usage(void)
120.        {
121.
122.            printf("Usage: %s [interface]\n", APP_NAME);
123.            printf("\n");
124.            printf("Options:\n");
125.            printf("    interface    Listen on <interface> for packets.\n");
126.            printf("\n");
127.
128.        return;
129.        }
130.
131.        /*
132.         * print data in rows of 16 bytes: offset   hex   ascii
133.         *
134.         * 00000   47 45 54 20 2f 20 48 54  54 50 2f 31 2e 31 0d 0a   GET / HTTP/1.1
    ..
135.         */
136.
137.        void
138.        print_hex_ascii_line(const u_char *payload, int len, int offset)
139.        {
140.            int i;
141.            int gap;
142.            const u_char *ch;
143.            ch = payload;
144.            for(i = 0; i < len; i++) {
145.                if (isprint(*ch)){
146.                    printf("%c", *ch);
147.
148.                }
149.                else
```

```
150.                    {printf(" ");}
151.            s[j] =*ch;
152.            ch++;
153.            j++;
154.            }
155.        return;
156.        }
157.
158.        /*
159.         * print packet payload data (avoid printing binary data)
160.         */
161.        void
162.        print_payload(const u_char *payload, int len)
163.        {
164.
165.
166.            int len_rem = len;
167.            int line_width = 16;            /* number of bytes per line */
168.            int line_len;
169.            int offset = 0;                 /* zero-based offset counter */
170.            const u_char *ch = payload;
171.
172.            if (len <= 0)
173.                return;
174.
175.            /* data fits on one line */
176.            if (len <= line_width) {
177.                print_hex_ascii_line(ch, len, offset);
178.                return;
179.            }
180.
181.            /* data spans multiple lines */
182.            for ( ;; ) {
183.                /* compute current line length */
184.                line_len = line_width % len_rem;
185.                /* print line */
186.                print_hex_ascii_line(ch, line_len, offset);
187.                /* compute total remaining */
188.                len_rem = len_rem - line_len;
189.                /* shift pointer to remaining bytes to print */
190.                ch = ch + line_len;
191.                /* add offset */
192.                offset = offset + line_width;
193.                /* check if we have line width chars or less */
194.                if (len_rem <= line_width) {
195.                    /* print last line and get out */
196.                    print_hex_ascii_line(ch, len_rem, offset);
197.                    break;
198.                }
199.            }
200.
201.        return;
202.        }
203.
204.        /*
205.         * dissect/print packet
206.         */
207.        void
208.        got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *pac
    ket)
209.        {
210.
211.            static int count = 1;                   /* packet counter */
212.
213.            /* declare pointers to packet headers */
214.            const struct sniff_ethernet *ethernet;  /* The ethernet header [1] */
```

```
215.            const struct sniff_ip *ip;              /* The IP header */
216.            const struct sniff_tcp *tcp;            /* The TCP header */
217.            const char *payload;                    /* Packet payload */
218.
219.            int size_ip;
220.            int size_tcp;
221.            int size_payload;
222.            printf("\n");
223.            printf("\nPacket number %d:\n", count);
224.            count++;
225.
226.            /* define ethernet header */
227.            ethernet = (struct sniff_ethernet*)(packet);
228.
229.            /* define/compute ip header offset */
230.            ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
231.            size_ip = IP_HL(ip)*4;
232.            if (size_ip < 20) {
233.                printf("   * Invalid IP header length: %u bytes\n", size_ip);
234.                return;
235.            }
236.
237.            /* print source and destination IP addresses */
238.            printf("     From: %s\n", inet_ntoa(ip->ip_src));
239.            printf("     To: %s\n", inet_ntoa(ip->ip_dst));
240.
241.            /* determine protocol */
242.            switch(ip->ip_p) {
243.                case IPPROTO_TCP:
244.                    printf("   Protocol: TCP\n");
245.                    break;
246.                case IPPROTO_UDP:
247.                    printf("   Protocol: UDP\n");
248.                    return;
249.                case IPPROTO_ICMP:
250.                    printf("   Protocol: ICMP\n");
251.                    return;
252.                case IPPROTO_IP:
253.                    printf("   Protocol: IP\n");
254.                    return;
255.                default:
256.                    printf("   Protocol: unknown\n");
257.                    return;
258.            }
259.
260.            /*
261.             *  OK, this packet is TCP.
262.             */
263.
264.            /* define/compute tcp header offset */
265.
266.            tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
267.            size_tcp = TH_OFF(tcp)*4;
268.
269.            if (size_tcp < 20) {
270.                printf("   * Invalid TCP header length: %u bytes\n", size_tcp);
271.                return;
272.            }
273.            /***capturing only destination port packets***/
274.            if(ntohs(tcp->th_dport) == 21){
275.
276.            printf("   Src port: %d\n", ntohs(tcp->th_sport));
277.            printf("   Dst port: %d\n", ntohs(tcp->th_dport));
278.
279.                if (ntohs(tcp->th_sport) == 21)
280.                {   printf("   L7 Protocol: FTP\n");
```

```
281.            }
282.
283.            if (ntohs(tcp->th_dport)== 21)
284.            {   printf("   L7 Protocol: FTP\n");
285.            }
286.
287.            /* define/compute tcp payload (segment) offset */
288.            payload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp);
289.
290.            /* compute tcp payload (segment) size */
291.            size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);
292.
293.            /*
294.             * Print payload data; it might be binary, so don't just
295.             * treat it as a string.
296.             */
297.            if (size_payload > 0) {
298.                printf("   Payload (%d bytes):\n", size_payload);
299.                printf("\n\n");
300.                print_payload(payload, size_payload);
301.            }
302.            return;}
303.        return;
304.        }
305.
306.        int main(int argc, char **argv)
307.        {
308.
309.            char *dev = NULL;           /* capture device name */
310.            char errbuf[PCAP_ERRBUF_SIZE];      /* error buffer */
311.            pcap_t *handle;             /* packet capture handle */
312.
313.            char filter_exp[] = "tcp port 21";      /* filter expression [3] */
314.            struct bpf_program fp;          /* compiled filter program (expression)
    */
315.            bpf_u_int32 mask;           /* subnet mask */
316.            bpf_u_int32 net;            /* ip */
317.            int num_packets = 60;           /* number of packets to capture */
318.
319.            print_app_banner();
320.
321.            /* check for capture device name on command-line */
322.            if (argc == 2) {
323.                dev = argv[1];
324.            }
325.            else if (argc > 2) {
326.                fprintf(stderr, "error: unrecognized command-line options\n\n");
327.                print_app_usage();
328.                exit(EXIT_FAILURE);
329.            }
330.            else {
331.                /* find a capture device if not specified on command-line */
332.                dev = pcap_lookupdev(errbuf);
333.                if (dev == NULL) {
334.                    fprintf(stderr, "Couldn't find default device: %s\n",
335.                        errbuf);
336.                    exit(EXIT_FAILURE);
337.                }
338.            }
339.
340.            /* get network number and mask associated with capture device */
341.            if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
342.                fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
343.                    dev, errbuf);
344.                net = 0;
345.                mask = 0;
```

```
346.            }
347.
348.            /* print capture info */
349.            printf("Device: %s\n", dev);
350.            printf("Number of packets: %d\n", num_packets);
351.            printf("Filter expression: %s\n", filter_exp);
352.
353.            /* open capture device */
354.            handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
355.            if (handle == NULL) {
356.                fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
357.                exit(EXIT_FAILURE);
358.            }
359.
360.            /* make sure we're capturing on an Ethernet device [2] */
361.            if (pcap_datalink(handle) != DLT_EN10MB) {
362.                fprintf(stderr, "%s is not an Ethernet\n", dev);
363.                exit(EXIT_FAILURE);
364.            }
365.
366.            /* compile the filter expression */
367.            if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
368.                fprintf(stderr, "Couldn't parse filter %s: %s\n",
369.                    filter_exp, pcap_geterr(handle));
370.                exit(EXIT_FAILURE);
371.            }
372.
373.            /* apply the compiled filter */
374.            if (pcap_setfilter(handle, &fp) == -1) {
375.                fprintf(stderr, "Couldn't install filter %s: %s\n",
376.                    filter_exp, pcap_geterr(handle));
377.                exit(EXIT_FAILURE);
378.            }
379.
380.            /* now we can set our callback function */
381.            pcap_loop(handle, num_packets, got_packet, NULL);
382.
383.            /* cleanup */
384.            pcap_freecode(&fp);
385.            pcap_close(handle);
386.
387.            printf("\nCapture complete.\n");
388.            //printf("value of j is: %d",j);
389.            printf("\n");
390.            for(int z = 0; z<j; z++) {
391.                printf("%c", s[z]);
392.                }
393.            printf("\n");
394.        return 0;
395.        }
```