Guruprasad Sivagurunatha Krishnan
107391284
sguru@iastate.edu

Program 7

```
#define RETSIGTYPE void
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <pcap.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#ifndef setsignal_h
#define setsignal_h
RETSIGTYPE (*setsignal(int, RETSIGTYPE (*)(int)))(int);
#endif
char cpre580f98[] = "netdump";
void raw_print(u_char *user, const struct pcap_pkthdr *h, const u_char *p);
int packettype;
char *program_name;
/* Externs */
extern void bpf_dump(struct bpf_program *, int);
extern char *copy_argv(char **);
/* Forwards */
void program_ending(int);
/* Length of saved portion of packet. */
int snaplen = 1500;;
static pcap_t *pd;
extern int optind;
extern int opterr;
extern char *optarg;
int pflag = 0, aflag = 0;
int arp_count = 0 ;
int ip_count = 0 ;
int broadcast_count = 0 ;
int icmp_count=0;
int tcp_count=0;
int dns_count=0;
int pop_count=0;
int smtp_count=0;
int imap_count=0;
int itemp;
int
main(int argc, char **argv)
{
int cnt, op, i, done = 0;
bpf_u_int32 localnet, netmask;
char *cp, *cmdbuf, *device;
struct bpf_program fcode;
void (*oldhandler)(int);
u_char *pcap_userdata;
char ebuf[PCAP_ERRBUF_SIZE];
cnt = -1;
device = NULL;
if ((cp = strrchr(argv[0], '/')) != NULL)
```

```
program_name = cp + 1;
else
program_name = argv[0];
opterr = 0;
while ((i = getopt(argc, argv, "pa")) != -1)
{
switch (i)
{
case 'p':
pflag = 1;
break;
case 'a':
aflag = 1;
break;
case '?':
default:
done = 1;
break;
}
if (done) break;
}
if (argc > (optind)) cmdbuf = copy_argv(&argv[optind]);
else cmdbuf = "";
if (device == NULL) {
device = pcap_lookupdev(ebuf);
if (device == NULL)
error("%s", ebuf);
}
pd = pcap_open_live(device, snaplen, 1, 1000, ebuf);
if (pd == NULL)
error("%s", ebuf);
i = pcap_snapshot(pd);
if (snaplen < i) {
warning("snaplen raised from %d to %d", snaplen, i);
snaplen = i;
}
if (pcap_lookupnet(device, &localnet, &netmask, ebuf) < 0) {
localnet = 0;
netmask = 0;
warning("%s", ebuf);
}
/*
* Let user own process after socket has been opened.
*/
setuid(getuid());
if (pcap_compile(pd, &fcode, cmdbuf, 1, netmask) < 0)
error("%s", pcap_geterr(pd));
(void)setsignal(SIGTERM, program_ending);
(void)setsignal(SIGINT, program_ending);
/* Cooperate with nohup(1) */
if ((oldhandler = setsignal(SIGHUP, program_ending)) != SIG_DFL)
(void)setsignal(SIGHUP, oldhandler);
if (pcap_setfilter(pd, &fcode) < 0)
error("%s", pcap_geterr(pd));
pcap_userdata = 0;
(void)fprintf(stderr, "%s: listening on %s\n", program_name, device);
if (pcap_loop(pd, cnt, raw_print, pcap_userdata) < 0) {
```

```
(void)fprintf(stderr, "%s: pcap_loop: %s\n",
program_name, pcap_geterr(pd));
exit(1);
}
pcap_close(pd);
exit(0);
}
/* routine is executed on exit */
void program_ending(int signo)
{
struct pcap_stat stat;
if (pd != NULL && pcap_file(pd) == NULL) {
(void)fflush(stdout);
putc('\n', stderr);
if (pcap_stats(pd, &stat) < 0)
(void)fprintf(stderr, "pcap_stats: %s\n",
pcap_geterr(pd));
else {
(void)fprintf(stderr, "%d packets received by filter\n",
stat.ps_recv);
(void)fprintf(stderr, "%d packets dropped by kernel\n",
stat.ps_drop);
(void)fprintf(stderr, "ARP packets: %d \n",
arp_count);
(void)fprintf(stderr, "IP packets: %d \n",
ip_count);
(void)fprintf(stderr, "Broadcast packets: %d \n",
broadcast_count);
(void)fprintf(stderr, "ICMP packets: %d \n",
icmp_count);
(void)fprintf(stderr, "No of TCP packets: %d \n",
tcp_count);
(void)fprintf(stderr, "No of DNS packets: %d \n",
dns_count);
(void)fprintf(stderr, "No of SMTP packets: %d \n",
smtp_count);
(void)fprintf(stderr, "No of POP packets: %d \n",
pop_count);
(void)fprintf(stderr, "No of IMAP packets: %d \n",
imap_count);
}
}
exit(0);
}
/* Like default_print() but data need not be aligned */
void
default_print_unaligned(register const u_char *cp, register u_int length)
{
register u_int i, s;
register int nshorts;
nshorts = (u_int) length / sizeof(u_short);
i = 0;
while (--nshorts >= 0) {
if ((i++ % 8) == 0)
(void)printf("\n\t\t\t");
s = *cp++;
(void)printf(" %02x%02x", s, *cp++);
```

```
}
if (length & 1) {
if ((i % 8) == 0)
(void)printf("\n\t\t\t");
(void)printf(" %02x", *cp);
}
}
/*
* By default, print the packet out in hex.
*/
void
default_print(register const u_char *bp, register u_int length)
{
register const u_short *sp;
register u_int i;
register int nshorts;
if ((long)bp & 1) {
default_print_unaligned(bp, length);
return;
}
sp = (u_short *)bp;
nshorts = (u_int) length / sizeof(u_short);
i = 0;
while (--nshorts >= 0) {
if ((i++ % 8) == 0)
(void)printf("\n\t");
(void)printf(" %04x", ntohs(*sp++));
}
if (length & 1) {
if ((i % 8) == 0)
(void)printf("\n\t");
(void)printf(" %02x", *(u_char *)sp);
}
}
/*
insert your code in this routine
*/
void raw_print(u_char *user, const struct pcap_pkthdr *h, const u_char *p)
{
int i;
int counter=0;
u_int length = h->len;
u_int caplen = h->caplen;
int upperbyte;
int lowerbyte;
int idpac;
int id;
int id1;
int lengthval;
counter = 0;
printf(" \n\n");
printf("__________________________________________________\n\n");
printf("\n\nDestination Hardware Address: ");
for(i=0;i<6;i++)
{
if(i==5)
{
```

```
if(p[i] == 0xFF)
counter++;
printf("%0x",p[i]);
printf("\n");
}
else
{
if(p[i]<0x10)
{
printf("%0x",0);
printf("%0x",p[i]);
printf(":");
}
else
{
if(p[i] == 0xFF)
counter++;
printf("%0x",p[i]);
printf(":");
}
}
}
if(counter==6)
broadcast_count++;
counter=0;
printf("Source MAC Address: ");
for(i=6;i<12;i++)
{
if(i==11)
{
printf("%0x",p[i]);
printf("\n");
}
else
{
if ( p[i] < 0x10)
{
printf("%0x",0);
printf("%0x",p[i]);
printf(":");
}
else
{
printf("%0x",p[i]);
printf(":");
}
}
}
//printf("\n");
printf("The Header Length: %0x \n",p[14]& 0x0F);
if ( p[12] == 0x08 && p[13] ==0x00)
{
printf("Packet Type: IP \n");
printf("IP version: %0x \n",p[14]>>4);
ip_count++;
printf("Header Length: %0x \n",p[14]& 0x0F);
printf("Type of Service: %0x \n",p[15]);
```

```
upperbyte=p[16];
upperbyte=upperbyte*256;
lowerbyte=p[17];
lengthval=upperbyte+lowerbyte;
printf("Length of the packet: %d \n",lengthval);
id= p[18];
id = id* 256;
id1 = p[19];
idpac = id+id1;
printf("Packet Id: %d \n",idpac);
printf("TTL is %d \n",p[22]);
int tcp_src_port;
int tcp_dst_port;
double tcp_seq_num;
long int ack_num;
int offset;
/* Handling TCP packets*/
if(p[23]== 0x06)
{
tcp_count++;
printf(" \nProtocol Number is 6 \n");
printf("\nPacket Type: TCP\n");
upperbyte=p[34]*256;
lowerbyte=p[35];
tcp_src_port=upperbyte+lowerbyte;
printf("Source Port: %d \n",tcp_src_port);
upperbyte=p[36]*256;
lowerbyte=p[37];
tcp_dst_port=upperbyte+lowerbyte;
//destination port
printf("Destination Port: %d\n",tcp_dst_port);
//count dns packets
if(tcp_src_port == 53 || tcp_dst_port == 53)
dns_count++;
//sequence number
printf("Sequence # : %lld\n",p[41]+(256*p[40])+(256*256*p[39])
+(256*256*256*p[38]));
//ack number
printf("ACK # : %lld\n",p[45]+(256*p[44])+(256*256*p[43])
+(256*256*256*p[42]));
//Data offset
printf("Offset: %d\n",p[47]+(256*p[46]));
if(tcp_src_port == 110 || tcp_dst_port == 110)
{
pop_count++;
printf(" \nPayload POP:\n\n ");
for(itemp=54;itemp<caplen;itemp++)
{
if(isprint(p[itemp])!=0)
printf("%c",p[itemp]);
}
printf(" \n\n === END OF POP Payload ===\n\n\n");
}
if(tcp_src_port == 143 || tcp_dst_port == 143)
{
imap_count++;
printf(" \nPayload IMAP:\n\n ");
```

```
for(itemp=54;itemp<caplen;itemp++)
{
if(isprint(p[itemp])!=0)
printf("%c",p[itemp]);
}
printf(" \n\n\n _____________________________\n\n\n");
}
if(tcp_src_port == 25 || tcp_dst_port == 25)
{
smtp_count++;
printf(" \nPayload SMTP:\n\n ");
for(itemp=54;itemp<caplen;itemp++)
{
if(isprint(p[itemp])!=0)
printf("%c",p[itemp]);
}
printf(" \n\n\n_____________________________\n\n\n");
}
if(tcp_src_port == 80 || tcp_dst_port == 80)
{
pop_count++;
printf(" \nHTTP Payload:\n\n ");
for(itemp=54;itemp<caplen;itemp++)
{
if(isprint(p[itemp])!=0)
printf("%c",p[itemp]);
}
printf(" \n\n ------- END OF HTTP Payload --------\n\n\n");
}
}
if(p[23]==0x01)
{
icmp_count++;
printf("The packet is an ICMP packet\nICMP Type : ",p[23]);
if (p[34]==0x00 && p[35]==0x00)
{
printf("Type: Request\n");
}
if (p[34]==0x04)
{
printf("Type: Source Quench\n");
}
if (p[34]==0x05)
{
printf("Type: Redirect\n");
}
if (p[34]==0x06)
{
printf("Type: Alternate Host Address\n");
}
if (p[34]==0x03)
{
printf("Type: ICMP Destination Unreachable\n");
}
if (p[34]==0x08 && p[35]==0x00)
{
printf("Type: Reply\n");
```

```
}
if (p[34]==0x09)
{
printf("Type: Router Adverstisment\n");
}
if (p[34]==10)
{
printf("Type: Router Selection\n");
}
if (p[34]==11)
{
printf("Type: Time Exceeded\n");
}
if (p[34]==12)
{
printf("Type: Parameter Problem\n");
}
if (p[34]==13)
{
printf("Type: Timestamp\n");
}
if (p[34]==14)
{
printf("Type: Timestamp reply\n");
}
}
if(p[23]==17)
{
printf("Packet Type: UDP \n");
}
printf("The Source IP: ");
for(i=26;i<30;i++)
{
if(i==29)
{
printf("%d",p[i]);
printf("\n");
}
else
{
printf("%d",p[i]);
printf(".");
}
}
printf("The Destination IP: ");
for(i=30;i<34;i++)
{
if(i==33)
{
printf("%d",p[i]);
printf("\n");
}
else
{
printf("%d",p[i]);
printf(".");
}
```

```
}
}
else if ( p[12] == 0x08 && p[13] ==0x06)
{
arp_count ++;
printf("ARP request/reply packet \n");
if(p[15] == 0x01)
{
printf("Hardware Type or Data Link Protocol is Ethernet\n");
}
if(p[21] == 0x01)
{
printf("ARP Request Packet\n");
}
if(p[21] == 0x02)
{
printf("ARP Reply Packet\n");
}
printf("");
printf("Hardware Address Length %d \n",p[18]);
printf("Length of Protocol Address %d \n",p[19]);
printf("Source MAC Address:\n");
printf("Sender IP: ");
for(i=28;i<32;i++)
{
if(i==31)
{
printf("%d",p[i]);
printf("\n");
}
else
{
printf("%d",p[i]);
printf(".");
}
}
printf("Destination Hardware Address: MAC\n");
printf("Destination IP: ");
for(i=38;i<42;i++)
{
if(i==41)
{
printf("%d",p[i]);
printf("\n");
}
else
{
printf("%d",p[i]);
printf(".");
}
}
}
// default_print(p, caplen);
putchar('\n');
}
```

OUTPUT:

```
destination IP address:129.186.96.141
TCP Header:
Source port:22
Destination port:35867
Sequence number:2685351510
Acknowledgement number:3376665286
Data offset:8 words
Reserved:0
Flags:CWR=0 ECE=0 URG=0 ACK=1 PSH=1 RST=0 SYN=0 FIN=0
Window:8326 bytes
Checksum:15441
Urgent pointer:0
Options:1 1 8 a c2 de 64 9a 36 aa fe 6d
000d 65d2 4ae0 0022 19ad 59d9 0800 4510
0134 ac74 4000 4006 5215 81ba d728 81ba
608d 0016 8c1b a00f 3656 c943 d2c6 8018
2086 3c51 0000 0101 080a c2de 649a 36aa
fe6d 8d7a e97e 5e9b 52ee fd64 e4d7 7ee1
7dd8 43d5 ad01 55ba f915 4594 2223 83bf
7283 f276 92f7 5901 38bb 60f5 c66a 2839
b047 f8e8 edd1 e184 8b7d fe84 6f61 87a1
41ab e118 7216 4de7 7a8d 4129 da7a 60e4
71f4 c34e 1e6c eace a884 0c11 fb70 cd24
edda 1500 ce49 daeb 71ca 6461 d4d5 279b
10e9 671f 25ee f7c6 d939 756a 9bdf 7c65
316c 9627 1e69 97c4 40bb 2bf2 db00 9c7b
ab84 ac29 12c3 6a29 5223 6787 554e 584e
5cfb a966 8cec ac00 fc04 26fb eeb1 6f9f
bdd7 19d2 7f13 63db f827 4fac 2b8f 98ab
80e5 f856 e666 81ac 9de4 c75e 3771 b284
23f2 3103 5545 087d 315e 61fe 4a15 d457
42ec b560 38a7 48c2 c030 b0dd 27af 1af3
ead4 5d8c e974 6fc6 3e77 dfbd bcef e4f6
c463
^CSource Ethernet address:0:22:19:ad:59:d9
202401 packets received by filter
153781 packets dropped by kernel
The number of IP packets = 48422
The number of ARP packets = 2
The number of broadcast packets = 13
The number of ICMP packets = 0
The number of TCP packets = 48370
The number of DNS packets = 2
The number of SMTP packets = 0
The number of POP packets = 0
The number of IMAP packets = 0
The number of HTTP packets = 34
```

HTTP Payload:

```
SLHTTP/1.0 200 OKServer: Apache-Coyote/1.1Content-Type:
text/html;charset=UTF-8Vary: Accept-EncodingContent-Encoding: gzipExpires:
Mon, 3 Dec 2012 18:38:24 GMTCache-Control: max-age=0, no-cache, no-
storePragma: nocacheDate:
Mon, 3 Dec 2012 18:38:24 GMTContent-Length: 6733Connection:
```

```
close\SH9TnfcY"ddr zxv,nVyn\Q>7g'q={{wYnSgaGwzo9<O^yt:mM[2z=kS{s
7kb}?Z@q#bW]""x2?
Dt\'>T)74KE,R7#yIePDl)sXT>"_GtyGnTd#A2fgvzsuqs{|",c5ras\q8NkY*<K/I9[\!$>?d;oJ
w,Q,
1"g(jR"9( :B[*)iXL&%0p79QbjHf-v3&/'( c&#[>9F]nMM;1C (!
$r6]14u5R6Um.d0'&HCW&Dq#&hdi1.*%!5]15|BZa#CJ5c8`Q.)>*/3qYAf$p.QMiD.Qt,
9<tu5b)Tafg@jp:pkXnE%N=wULTdq2<1X-D^nt5z2)z7Z;][[i??*"cTP@(pU=;$>,t\AQ+p2rgo
$c7_Et4[8vXp%X\|= D$|t7tn/m^?wsU^eQ{{>T~a V=YkM">)OVEhA]n'0Z
--------END of HTTP Payload----------
0022 19ad 59d9 000d 65d2 4ae0 0800 4500
05dc 1067 4000 3a06 8481 d138 7c18 81ba
d728 0050 f6b0 760c b40b 0450 9899 8010
0c90 0782 0000 0101 080a ede8 b1df 8953
a34c 4854 5450 2f31 2e30 2032 3030 204f
4b0d 0a53 6572 7665 723a 2041 7061 6368
652d 436f 796f 7465 2f31 2e31 0d0a 436f
6e74 656e 742d 5479 7065 3a20 7465 7874
2f68 746d 6c3b 6368 6172 7365 743d 5554
462d 380d 0a56 6172 793a 2041 6363 6570
742d 456e 636f 6469 6e67 0d0a 436f 6e74
656e 742d 456e 636f 6469 6e67 3a20 677a
6970 0d0a 4578 7069 7265 733a 2046 7269
2c20 3138 2044 6563 2032 3030 3920 3230
3a35 363a 3536 2047 4d54 0d0a 4361 6368
652d 436f 6e74 726f 6c3a 206d 6178 2d61
6765 3d30 2c20 6e6f 2d63 6163 6865 2c20
6e6f 2d73 746f 7265 0d0a 5072 6167 6d61
3a20 6e6f 2d63 6163 6865 0d0a 4461 7465
3a20 4672 692c 2031 3820 4465 6320 3230
3039 2032 303a 3536 3a35 3620 474d 540d
0a43 6f6e 7465 6e74 2d4c 656e 6774 683a
2036 3733 330d 0a43 6f6e 6e65 6374 696f
6e3a 2063 6c6f 7365 0d0a 0d0a 1f8b 0800
0000 0000 0000 ed5c fb53 db48 b6fe 3954
cdff d0d1 6e0d 6663 5918 088f 809d 2284
64b8 9700 0b64 e6ee 9ddd 72b5 a5b6 ad20
abb5 7ad8 7876 e67f bfdf e987 2cd9 8690
c9cc 6ed5 decd 03a4 56f7 79f5 e9af cfe9
6ee9 e8f9 dbcb 93db bf5c 9db2 513e 8ed8
d5c7 37e7 6727 cc71 3def 87ed 13cf 7b7b
fb96 fdcf 77b7 1fce 59bb b5c9 6e53 1e67
611e ca98 479e 777a e1b0 6fd6 9e39 a33c
4f5e 79de 743a 6d4d b75b 321d 7ab7 d7de
3d91 6b53 7b73 e9e6 95c6 ad20 0f9c ee37
6bfa ef91 627d 3f8e e2ac b382 5afb e0e0
4013 71a8 d2ab 88c7 c38e 2362 8795 575d
2222 7880 dfcf 8e32 3f0d 939c e5b3 4474
9c5c dce7 de27 3ee1 ba54 b729 f810 8ffe
0ba5 37a6 344b fd8e e30d 452c 529e 8ba0
3708 2391 7949 d1ef 8d65 5044 c2d3 cdbd
6cb7 f529 7358 f7c8 1454 193e 99c6 f68e
22f2 8080 5f47 bbbd bff7 74e2 7998 47a2
cb6e 8a54 6423 f641 c432 66ec 6776 7a73
7571 737b 7cdd f2e5 f8c8 d395 d68e c622
```