

SNIFFER.C Dokumentace

OBSAH

1.Popis kódu

2.Testovanie

Popis kódu

Na začiatku môjho zdrojového kódu môžete vidieť štruktúry, ktoré som vytvoril podľa skopírovaných štruktúr sniff_ethernet, sniff_ip, a sniff_tcp.

```
//kod prevzatý a upravený z
//https://www.tcpdump.org/pcap.html
/* 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 /* 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 */
};

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

Po skopírovaní štruktúr som pochopil princíp ako previesť hlavičky jednotlivých protokolov na dátovú štruktúru.

```
struct IPv4_address{
    u_int8_t first_part;
    u_int8_t second_part;
    u_int8_t third_part;
    u_int8_t fourth_part;
};

struct sniff_udp {
    u_short udp_sport; /* source port */
    u_short udp_dport; /* destination port */
    u_short udp_len; /*udp len*/
    u_short udp_sum; /* checksum */
};

struct sniff_icmp {
    u_char icmp_type; /* type */
    u_char icmp_code; /* error code */
    u_short icmp_checksum; /*checksum */
};
```

Tieto štruktúry patria k najdôležitejším častiam projektu.

Ďalej som pomocou cyklu while a switchu spracoval jednotlivé parametry.

Switch mi pomohol pri vyhodnotení krátkych argumentov ako napr. -i eth0

```
while((short_option=getopt_long(argc,argv,opt_string,options,&opt_index))!=-1)
{
    switch (short_option) //find out options and stores them
    {
        case 'i':
            interface_set=true;
            if(argv[optind]!=NULL)
            {
                if(argv[optind][0]!='-')// think about something better what if --- somebody insert 3-
                {
                    interface=argv[optind];
                }
            }
        }
    }
```

Ak sa jednalo o možnosti ktoré nmali krátku verzie tak som vedel že getopt pre nich vráť ? a keďže boli iba 2 tak som ich následne spracoval pomocou podmienky if.

```

default: //if it is long option
    if(strcmp("arp",options[opt_index].name)==0)
    {
        protocols[ARP]=true;
    }
    else if (strcmp("icmp",options[opt_index].name)==0)
    {
        protocols[ICMP]=true;
    }
    else
    {
        error_input();
    }

```

Po zpracovaní argumentov som pomocou funkcie `create_filter` expression ktorá triviálne vytvára string pomocou skupiny podmienok `if else`. Vytvorený string obsahuje parametry pre filter.

Následne program pomocou tohto kódu prevzatého z

<https://www.tcpdump.org/pcap.html>

Otvoril interface na čítanie a nastavil filter na filtrovanie požadovaných framov.

<https://www.tcpdump.org/pcap.html>

```

//kod prevzatý a upravený zo stránky
//https://www.tcpdump.org/pcap.html
pcap_t *handle; //frame handle
char *filter_exp; //filter_exp
filter_exp=malloc(sizeof(char));
create_filter_expression(filter_exp,protocols,port_number);//create expression for filter
struct pcap_pkthdr header; /* The header that pcap gives us */
bpf_u_int32 mask; /* The netmask of our sniffing device */
bpf_u_int32 net; /* The IP of our sniffing device */
// const u_char *packet;
struct bpf_program fp;
if (pcap_lookupnet(interface, &net, &mask, errbuf) == -1) {
    fprintf(stderr, "Can't get netmask for device %s\n", interface);
    net = 0;
    mask = 0;
}
handle = pcap_open_live(interface, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
    fprintf(stderr, "Couldn't open device %s: %s\n", interface, errbuf);
    return(2);
}
if (pcap_datalink(handle) != DLT_EN10MB) { //check if it is ethernet header
    fprintf(stderr, "Device %s doesn't provide Ethernet headers - not supported\n", interface);
    return(2);
}
int pcap_compile()
if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
    fprintf(stderr, "Couldn't parse filter %s: %s\n", filter_exp, pcap_geterr(handle));
    return(2);
}
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Couldn't install filter %s: %s\n", filter_exp, pcap_geterr(handle));
    return(2);
}
int a;
a=pcap_loop(handle,number_of_packets,got_packet,NULL);
//konec citace

```

Program ďalej pomocou funkcie got_packet vypisuje packety na výstup.

Funkcia postupne zbavuje dáta jednotlivých hlavičiek a dáta hlavičiek ukladá do štruktúr.

`ethernet = (struct sniff_ethernet*)(packet);` začíname s ethernetovou hlavičkou ktorú priradíme do štruktúry ktorá bude začínať tam kde ukazuje packet a bude končiť na mieste packet + veľkosť ethernetu. Tým že jednotlivé dáta máme v štruktúre tak s nimi dokážeme efektívne pracovať.

`ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);` To isté spravíme s IP hlavičkou. IP hlavička začína tam kde končí ethernetová hlavička.

Ďalej program vyberie správny protokol podľa hodnoty v štruktúre ip. Napr v IP hlavičke bol nasledujúci protokol TCP

```
if(ip->ip_p==TCP_PROTOCOL) //tcp
{
    tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
    size_tcp = TH_OFF(tcp);
    size_tcp =size_tcp*4;
    if (size_tcp < 20) {
        printf(" * Invalid TCP header length: %u bytes\n", size_tcp);
        return;
    }
    u_int16_t src_port=ntohs(tcp->th_sport);
    printf("src_prt: %d\n",src_port);
    u_int16_t dst_port=ntohs(tcp->th_dport);
    printf("dst_prt: %d\n",dst_port);
    payload = (u_char *) (packet + SIZE_ETHERNET + size_ip + size_tcp);
    p_header_size=size_tcp;
}
```

Na začiatku funkcie môžeme vydieť že do tcp sa opäť uloží ukazateľ tenko krát na štruktúru sniff_tcp. Zistíme veľkosť TCP.

Ďalej program načíta preusporiada biti do správneho poradia a vypíše hodnoty pre port príjmateľa a odosielateľa. Payload alebo dáta je časť packeta kde sa nachádzajú dáta packetu tie potom následne program vypíše pomocou cyklu for(hexa) a vnoreného cyklu(ascii)

```

for (int j=0;j<packet_len-(size_ip+p_header_size);j++)
{
    if(j==0)
    {
        printf("0x%04x\t",offset);
    }
    u_char hex=payload[j];
    printf("%02x ",hex);
    if((chars_counter==15) || (j+1)==packet_len-(size_ip+p_header_size))
    {
        printf("\t");
        int chars2_counter=0;
        for(int i=j-chars_counter;i<packet_len-(size_ip+p_header_size);i++)
        {
            if(payload[i]>=32 && payload[i]<=126)
            {
                printf("%c",payload[i]);
            }
            else
            {
                printf(".");
            }
            if(chars2_counter==chars_counter)
            {
                break;
            }
            chars2_counter++;
        }
        offset=offset+16;
        printf("\n");
        if((j+1)!=packet_len-(size_ip+p_header_size))
        {
            printf("0x%04x\t",offset);
            chars_counter=0;
        }
        continue;
    }
    chars_counter++;
}

```

Cyklus ide do vtedy od aktuálnej pozície nakoniec dát. Hodnotu získame tak že od celkovej veľkosti packetu odpočítame jednotlivé veľkosti hlavičiek. Následne sa vypíše znak hexadecimálne. No konci cyklu máme counter ktorý nám ráta koľko znakov bolo vypísaných. Ak bolo vypísaných 16 znakov alebo program sa distak nakoniec dát. Začneme vypisovať znaky ascii ktorých tiež vypíšeme 16 a posuniem sa na nový riadok.

Podobne program funguje aj pre protokol IPv6.

Testovanie

Projekt som testoval vo Wireshark a hodnoty som porovnal s hodnotami môjho programu pridávam screenshot.

```
Timestamp: 2022-04-24T22:32:50.976+01:00
src MAC: 08:95:9f:69:e8:6c
dst MAC: ff:ff:ff:ff:ff:ff
src IP: 100.64.196.28
dst IP: 100.64.223.255
src_port: 54915
dst_port: 54915

0x0000 00 4d 53 49 00 00 00 00 00 78 76 0d 8d 02 00 00 .....PSI.....xv.....
0x0010 80 b9 1f 6e d5 00 00 00 00 00 78 76 0d 8d 02 00 00 .....3'.....xv.....
0x0020 33 27 00 00 00 00 00 00 90 78 76 0d 8d 02 00 00 .....0.....
0x0030 f0 4b 40 0d 02 00 00 00 00 b7 1c 0e 0d 02 00 00 .....|ja.....
0x0040 00 00 00 00 00 00 00 00 7c 6a 61 5e 00 00 00 00 .....U.....
0x0050 80 a5 55 5e 00 00 00 00 b9 bd 1f 6e d5 00 00 00 .....0.....
0x0060 00 00 00 00 00 00 00 00 93 40 0d 8d 02 00 00 .....0.....
0x0070 64 ba 1f 6e d5 00 00 00 20 ba 1f 6e d5 00 00 00 .....N.....
0x0080 08 f9 4e 7b 33 33 66 32 36 30 63 35 2d 64 65 38 .....N(33f260c5-de8
0x0090 39 2d 34 37 36 30 2d 61 36 35 30 2d 39 32 65 63 .....9-4760-a650-92ec
0x00a0 38 23 38 32 31 66 36 36 7d 00 00 00 00 00 00 00 .....83821f0e).....
0x00b0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0100 00 00 00 9e 2a 8a bd .....*.,.

Timestamp: 2022-04-24T22:32:51.888+01:00
src MAC: c0:3c:59:35:e3:19
dst MAC: ff:ff:ff:ff:ff:ff
src IP: 100.64.282.36
dst IP: 100.64.223.255
src_port: 54915
dst_port: 54915

0x0000 00 4c 41 50 54 4f 50 2d 52 4e 46 37 41 38 56 42 .....LAPTOP-RNF7ABV8
0x0010 00 00 00 00 00 00 00 00 b0 d3 17 78 ab 01 00 00 .....0.....Ts.....
0x0020 10 bc af 1d 30 00 00 00 00 54 73 ab 01 00 00 .....3.....X.....
0x0030 33 27 00 00 00 00 00 00 a0 d3 17 78 ab 01 00 00 .....ns.....t.....
0x0040 f0 6d 6d 73 ab 01 00 00 a0 70 d7 76 ab 01 00 00 .....0.....t.....
0x0050 d0 bf af 1d 30 00 00 00 c6 74 92 7c fe 7f 00 00 .....0.....
0x0060 07 e1 00 00 00 00 00 f0 bd af 1d 30 00 00 00 .....h.....m.....
0x0070 a0 68 c8 75 ab 01 00 00 00 00 00 00 00 00 00 00 .....Z.....
0x0080 98 5a 13 76 34 30 62 63 32 36 64 65 2d 31 39 30 .....a-452b-8e2e-4dc4
0x0090 61 2d 34 35 32 62 2d 38 65 32 65 2d 34 64 63 34 .....e7b0fad4).Ls.....
0x00a0 65 27 62 30 66 61 64 34 7d 00 6c 73 ab 01 00 00 .....
0x00b0 f0 bd af 1d 1c 00 00 00 00 00 00 00 00 00 00 00 .....
0x00c0 01 00 00 00 00 00 00 90 bc af 1d 30 00 00 00 .....0.....
0x00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00f0 a0 8c 2d 35 94 d2 00 00 8a 51 1b 73 fe 7f 00 00 .....5.....Q.....
0x0100 07 01 00 0b 76 0c 04 .....v.,.
```

