# 网络安全实践课程实验报告

网络嗅探器

张一白

161320224

目录

[网络安全实践课程实验报告 1](#_Toc454485221)

[1. 实验要求 3](#_Toc454485222)

[2. 实验环境 3](#_Toc454485223)

[3. 系统设计与实现 3](#_Toc454485224)

[4. 使用说明 5](#_Toc454485225)

[5. 程序运行截图 5](#_Toc454485226)

[6. 源代码 5](#_Toc454485227)

## 实验要求

采用socket设计并实现一个网络嗅探器，能够抓包并对包进行解析。

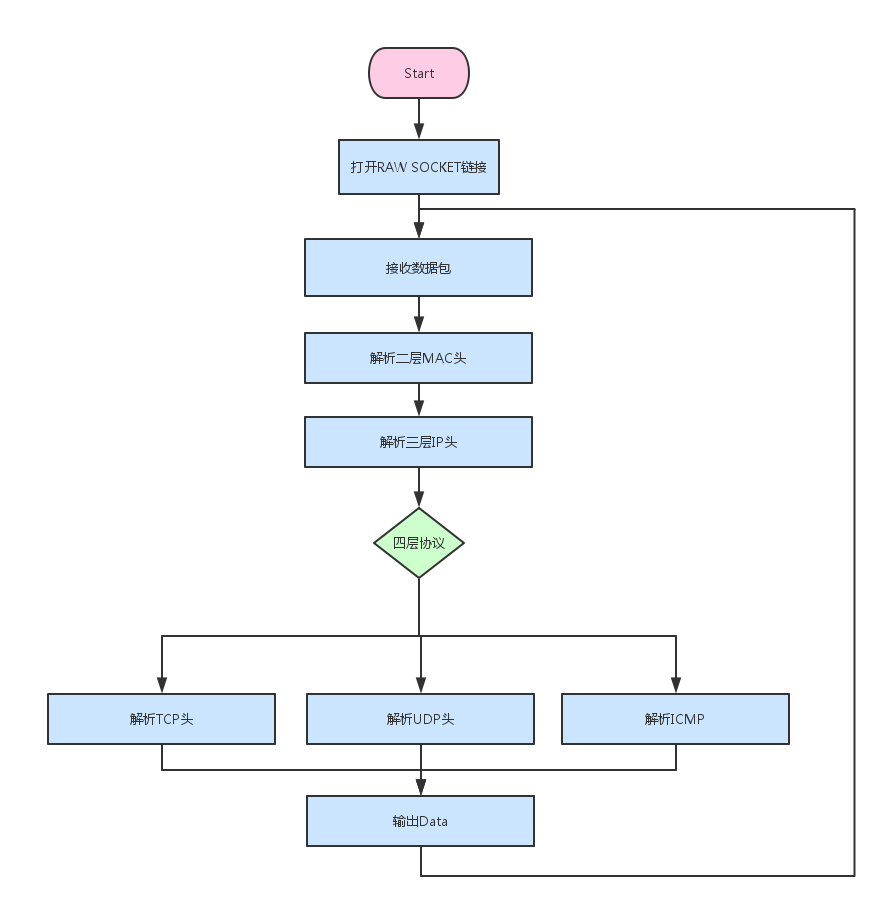
## 实验环境

Linux Ubuntu-Dev 4.4.0-24-generic #43-Ubuntu SMP Wed Jun 8 19:27:37 UTC 2016 x86\_64 x86\_64 x86\_64 GNU/Linux

## 系统设计与实现

（重点流程图+说明）

采用RAW SOCKET方式获取二层数据包，随后拆开数据链路层输出双方mac地址，并将payload传入下一层拆包拆出IP头和IP层payload。随后根据IP头中解析出来的Protocol选择合适的协议解析过程进行具体的解析。所有的解析数据存入log.txt



## 使用说明

编译：

gcc main.c –o sniffer –O2 (Linux with gcc)

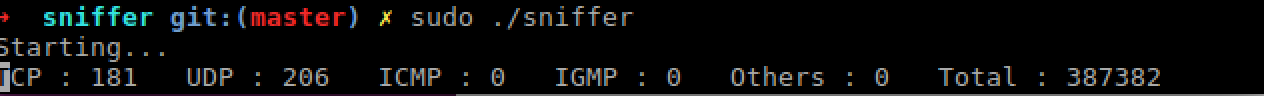
运行:

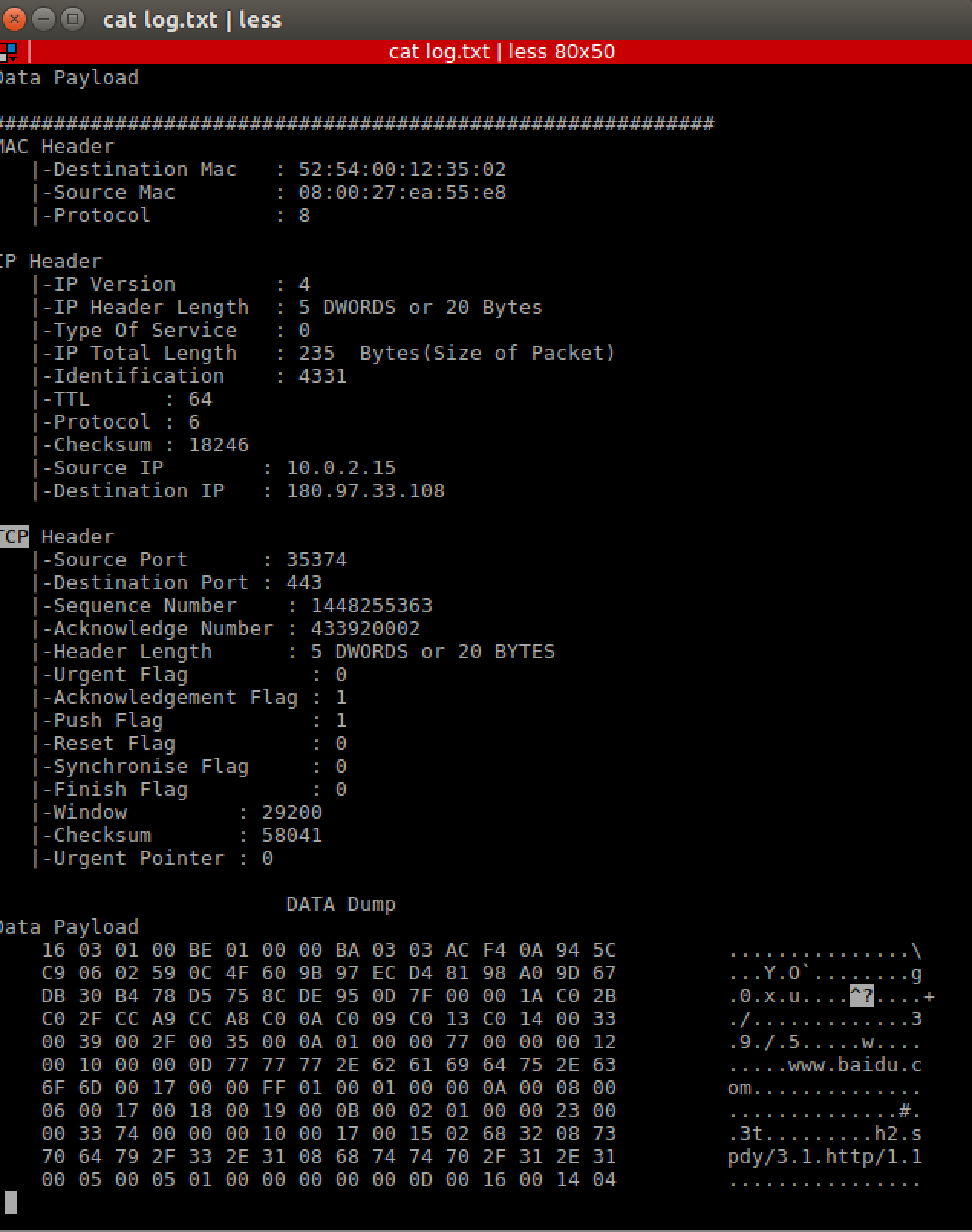
sudo ./sniffer

或以root身份运行

./sniffer

## 程序运行截图





## 源代码

|  |
| --- |
| //  // main.c  // sniffer  //  // Created by Summer on 6/9/16.  // Copyright © 2016 summer. All rights reserved.  //  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <errno.h>  #include <netinet/ip\_icmp.h>  #include <netinet/udp.h>  #include <netinet/tcp.h>  #include <netinet/ip.h>  #include <net/ethernet.h>  #include <sys/socket.h>  #include <arpa/inet.h>  #include <linux/if\_ether.h>  void ProcessPacket(unsigned char \*, int);  void print\_ether\_header(unsigned char \*buffer, int size);  void print\_ip\_header(unsigned char \*, int);  void print\_tcp\_packet(unsigned char \*, int);  void print\_udp\_packet(unsigned char \*, int);  void print\_icmp\_packet(unsigned char \*, int);  void PrintData(unsigned char \*, int);  int sock\_raw;  FILE \*logfile;  int tcp = 0, udp = 0, icmp = 0, others = 0, igmp = 0, total = 0, i, j;  struct sockaddr\_in source, dest;  int main() {  int saddr\_size, data\_size;  struct sockaddr saddr;  struct in\_addr in;  unsigned char \*buffer = (unsigned char \*) malloc(2048);  logfile = fopen("log.txt", "w");  if (logfile == NULL)  printf("Unable to create file.");  printf("Starting...\n");    sock\_raw = socket(AF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ALL));  if (sock\_raw < 0) {  printf("Socket Error: %s\n", strerror(errno));  return 1;  }  while (1) {  saddr\_size = sizeof saddr;    data\_size = recvfrom(sock\_raw, buffer, 2048, 0, &saddr, (socklen\_t \*) &saddr\_size);  if (data\_size < 0) {  printf("Recvfrom error , failed to get packets\n");  return 1;  }    ProcessPacket(buffer, data\_size);  }  close(sock\_raw);  printf("Finished");  return 0;  }  void ProcessPacket(unsigned char \*buffer, int size) {  print\_ether\_header(buffer, size);  buffer = buffer + sizeof(struct ether\_header);  size = size - sizeof(struct ether\_header);    print\_ip\_header(buffer, size);  struct iphdr \*iph = (struct iphdr \*) buffer;  ++total;  switch (iph->protocol)  {  case 1:  ++icmp;  print\_icmp\_packet(buffer, size);  break;  case 2:  ++igmp;  break;  case 6:  ++tcp;  print\_tcp\_packet(buffer, size);  break;  case 17:  ++udp;  print\_udp\_packet(buffer, size);  break;  default:  ++others;  break;  }  printf("TCP : %d UDP : %d ICMP : %d IGMP : %d Others : %d Total : %d\r", tcp, udp, icmp, igmp, others,  total);  fprintf(logfile, "\n###########################################################");  fflush(logfile);  fflush(stdout);  }  void print\_ether\_header(unsigned char \*buffer, int size) {  struct ether\_header \*ethhdr = (struct ether\_header \*) buffer;  fprintf(logfile, "\n");  fprintf(logfile, "MAC Header\n");  fprintf(logfile, " |-Destination Mac : %02x:%02x:%02x:%02x:%02x:%02x\n",  ethhdr->ether\_dhost[0], ethhdr->ether\_dhost[1], ethhdr->ether\_dhost[2], ethhdr->ether\_dhost[3],  ethhdr->ether\_dhost[4], ethhdr->ether\_dhost[5]);  fprintf(logfile, " |-Source Mac : %02x:%02x:%02x:%02x:%02x:%02x\n",  ethhdr->ether\_shost[0], ethhdr->ether\_shost[1], ethhdr->ether\_shost[2], ethhdr->ether\_shost[3],  ethhdr->ether\_shost[4], ethhdr->ether\_shost[5]);  fprintf(logfile, " |-Protocol : %d\n", ethhdr->ether\_type);  }  void print\_ip\_header(unsigned char \*Buffer, int Size) {  unsigned short iphdrlen;  struct iphdr \*iph = (struct iphdr \*) Buffer;  iphdrlen = iph->ihl \* 4;  memset(&source, 0, sizeof(source));  source.sin\_addr.s\_addr = iph->saddr;  memset(&dest, 0, sizeof(dest));  dest.sin\_addr.s\_addr = iph->daddr;  fprintf(logfile, "\n");  fprintf(logfile, "IP Header\n");  fprintf(logfile, " |-IP Version : %d\n", (unsigned int) iph->version);  fprintf(logfile, " |-IP Header Length : %d DWORDS or %d Bytes\n", (unsigned int) iph->ihl,  ((unsigned int) (iph->ihl)) \* 4);  fprintf(logfile, " |-Type Of Service : %d\n", (unsigned int) iph->tos);  fprintf(logfile, " |-IP Total Length : %d Bytes(Size of Packet)\n", ntohs(iph->tot\_len));  fprintf(logfile, " |-Identification : %d\n", ntohs(iph->id));  fprintf(logfile, " |-TTL : %d\n", (unsigned int) iph->ttl);  fprintf(logfile, " |-Protocol : %d\n", (unsigned int) iph->protocol);  fprintf(logfile, " |-Checksum : %d\n", ntohs(iph->check));  fprintf(logfile, " |-Source IP : %s\n", inet\_ntoa(source.sin\_addr));  fprintf(logfile, " |-Destination IP : %s\n", inet\_ntoa(dest.sin\_addr));  }  void print\_tcp\_packet(unsigned char \*Buffer, int Size) {  unsigned short iphdrlen;  struct iphdr \*iph = (struct iphdr \*) Buffer;  iphdrlen = iph->ihl \* 4;  struct tcphdr \*tcph = (struct tcphdr \*) (Buffer + iphdrlen);  fprintf(logfile, "\n");  fprintf(logfile, "TCP Header\n");  fprintf(logfile, " |-Source Port : %u\n", ntohs(tcph->source));  fprintf(logfile, " |-Destination Port : %u\n", ntohs(tcph->dest));  fprintf(logfile, " |-Sequence Number : %u\n", ntohl(tcph->seq));  fprintf(logfile, " |-Acknowledge Number : %u\n", ntohl(tcph->ack\_seq));  fprintf(logfile, " |-Header Length : %d DWORDS or %d BYTES\n", (unsigned int) tcph->doff,  (unsigned int) tcph->doff \* 4);  fprintf(logfile, " |-Urgent Flag : %d\n", (unsigned int) tcph->urg);  fprintf(logfile, " |-Acknowledgement Flag : %d\n", (unsigned int) tcph->ack);  fprintf(logfile, " |-Push Flag : %d\n", (unsigned int) tcph->psh);  fprintf(logfile, " |-Reset Flag : %d\n", (unsigned int) tcph->rst);  fprintf(logfile, " |-Synchronise Flag : %d\n", (unsigned int) tcph->syn);  fprintf(logfile, " |-Finish Flag : %d\n", (unsigned int) tcph->fin);  fprintf(logfile, " |-Window : %d\n", ntohs(tcph->window));  fprintf(logfile, " |-Checksum : %d\n", ntohs(tcph->check));  fprintf(logfile, " |-Urgent Pointer : %d\n", tcph->urg\_ptr);  fprintf(logfile, "\n");  fprintf(logfile, " DATA Dump ");  fprintf(logfile, "\n");  fprintf(logfile, "Data Payload\n");  PrintData(Buffer + iphdrlen + tcph->doff \* 4, (Size - tcph->doff \* 4 - iph->ihl \* 4));  }  void print\_udp\_packet(unsigned char \*Buffer, int Size) {  unsigned short iphdrlen;  struct iphdr \*iph = (struct iphdr \*) Buffer;  iphdrlen = iph->ihl \* 4;  struct udphdr \*udph = (struct udphdr \*) (Buffer + iphdrlen);  fprintf(logfile, "\nUDP Header\n");  fprintf(logfile, " |-Source Port : %d\n", ntohs(udph->source));  fprintf(logfile, " |-Destination Port : %d\n", ntohs(udph->dest));  fprintf(logfile, " |-UDP Length : %d\n", ntohs(udph->len));  fprintf(logfile, " |-UDP Checksum : %d\n", ntohs(udph->check));  fprintf(logfile, "Data Payload\n");  PrintData(Buffer + iphdrlen + sizeof udph, (Size - sizeof udph - iph->ihl \* 4));  }  void print\_icmp\_packet(unsigned char \*Buffer, int Size) {  unsigned short iphdrlen;  struct iphdr \*iph = (struct iphdr \*) Buffer;  iphdrlen = iph->ihl \* 4;  struct icmphdr \*icmph = (struct icmphdr \*) (Buffer + iphdrlen);  fprintf(logfile, "\nICMP Header\n");  fprintf(logfile, " |-Type : %d", (unsigned int) (icmph->type));  if ((unsigned int) (icmph->type) == 11)  fprintf(logfile, " (TTL Expired)\n");  else if ((unsigned int) (icmph->type) == ICMP\_ECHOREPLY)  fprintf(logfile, " (ICMP Echo Reply)\n");  fprintf(logfile, " |-Code : %d\n", (unsigned int) (icmph->code));  fprintf(logfile, " |-Checksum : %d\n", ntohs(icmph->checksum));  fprintf(logfile, "\n");  fprintf(logfile, "Data Payload\n");  PrintData(Buffer + iphdrlen + sizeof icmph, (Size - sizeof icmph - iph->ihl \* 4));  }  void PrintData(unsigned char \*data, int Size) {  for (i = 0; i < Size; i++) {  if (i != 0 && i % 16 == 0)  {  fprintf(logfile, " ");  for (j = i - 16; j < i; j++) {  if (data[j] >= 32 && data[j] <= 128)  fprintf(logfile, "%c", (unsigned char) data[j]);  else fprintf(logfile, ".");  }  fprintf(logfile, "\n");  }  if (i % 16 == 0) fprintf(logfile, " ");  fprintf(logfile, " %02X", (unsigned int) data[i]);  if (i == Size - 1)  {  for (j = 0; j < 15 - i % 16; j++) fprintf(logfile, " ");  fprintf(logfile, " ");  for (j = i - i % 16; j <= i; j++) {  if (data[j] > 31 && data[j] < 128) fprintf(logfile, "%c", (unsigned char) data[j]);  else fprintf(logfile, ".");  }  fprintf(logfile, "\n");  }  }  } |