网络编程第三次作业

ping 源码撰写 & 运行

孙利峰
57119218

2022 年 9 月 27 日

1 实验经历

本次实验没有难点，原因在于老师给的代码在 Ubuntu 系统上能够完整运行，只需要使用 gcc 编译即可。观察源代码 main 函数部分可以发现设置了 setuid，所以可以通过 chmod 将其设为 4 来设置 setuid 位。

2 代码运行逻辑

- 检查输入参数是否正确
- 检查输入的是 ip 还是域名，并检查域名是否合法
- 构造数据包与 ICMP& IP 报头
- 发送原始报文
- 接收报文并拆包
- 分析

3 运行结果

由于我使用的是 wsl 模拟 Ubuntu22.04，而 wsl 的某些特性，导致这里无法设置 setuid，只能使用 sudo 进行运行。

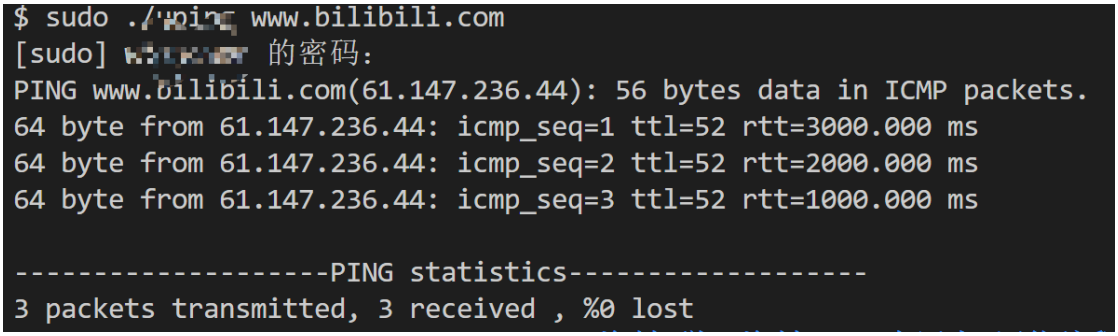

图 1: 运行结果

代码:

```
#include <stdio.h>
#include <signal.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <setjmp.h>
#include <errno.h>

#define PACKET_SIZE 4096
#define MAX_WAIT_TIME 5
#define MAX_NO_PACKETS 3

char sendpacket[PACKET_SIZE];
char recvpacket[PACKET_SIZE];
int sockfd, datalen = 56;
int nsend = 0, nreceived = 0;
struct sockaddr_in dest_addr;
pid_t pid;
```

```
struct sockaddr_in from;
struct timeval tvrecv;

void statistics(int signo);
unsigned short cal_chksum(unsigned short *addr, int len);
int pack(int pack_no);
void send_packet(void);
void recv_packet(void);
int unpack(char *buf, int len);
void tv_sub(struct timeval *out, struct timeval *in);

void statistics(int signo)
{
    printf("\n--------------------PING statistics-------------------\n");
    printf("%d packets transmitted, %d received , %%%d lost\n", nsend, nreceived,
           (nsend - nreceived) / nsend * 100);
    close(sockfd);
    exit(1);
}
/*校验和算法*/
unsigned short cal_chksum(unsigned short *addr, int len)
{
    int nleft = len;
    int sum = 0;
    unsigned short *w = addr;
    unsigned short answer = 0;

    /*把ICMP报头二进制数据以2字节为单位累加起来*/
    while (nleft > 1)
    {
        sum += *w++;
        nleft -= 2;
    }
    /*若ICMP报头为奇数个字节，会剩下最后一字节。
    把最后一个字节视为一个2字节数据的高字节，这个2字节数据的低字节为0，继续累加*/
    if (nleft == 1)
```

```
    {
        *(unsigned char *)(&answer) = *(unsigned char *)w;
        sum += answer;
    }
    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    answer = ~sum;
    return answer;
}
/*设置ICMP报头*/
int pack(int pack_no)
{
    int i, packsize;
    struct icmp *icmp;
    struct timeval *tval;

    icmp = (struct icmp *)sendpacket;
    icmp->icmp_type = ICMP_ECHO;
    icmp->icmp_code = 0;
    icmp->icmp_cksum = 0;
    icmp->icmp_seq = pack_no;
    icmp->icmp_id = pid;
    packsize = 8 + datalen;
    tval = (struct timeval *)icmp->icmp_data;
    gettimeofday(tval, NULL);                                          /*记录发送时间*/
    icmp->icmp_cksum = cal_chksum((unsigned short *)icmp, packsize); /*校验算法*/
    return packsize;
}

/*发送三个ICMP报文*/
void send_packet()
{
    int packetsize;
    while (nsend < MAX_NO_PACKETS)
    {
        nsend++;
```

```
        packetsize = pack(nsend); /*设置ICMP报头*/
        if (sendto(sockfd, sendpacket, packetsize, 0,
                   (struct sockaddr *)&dest_addr, sizeof(dest_addr)) < 0)
        {
            perror("sendto error");
            continue;
        }
        sleep(1); /*每隔一秒发送一个ICMP报文*/
    }
}

/*接收所有ICMP报文*/
void recv_packet()
{
    int n, fromlen;
    extern int errno;

    signal(SIGALRM, statistics);
    fromlen = sizeof(from);
    while (nreceived < nsend)
    {
        alarm(MAX_WAIT_TIME);
        if ((n = recvfrom(sockfd, recvpacket, sizeof(recvpacket), 0,
                          (struct sockaddr *)&from, &fromlen)) < 0)
        {
            if (errno == EINTR)
                continue;
            perror("recvfrom error");
            continue;
        }
        gettimeofday(&tvrecv, NULL); /*记录接收时间*/
        if (unpack(recvpacket, n) == -1)
            continue;
        nreceived++;
    }
}
```

```
/*剥去ICMP报头*/
int unpack(char *buf, int len)
{
    int i, iphdrlen;
    struct ip *ip;
    struct icmp *icmp;
    struct timeval *tvsend;
    double rtt;

    ip = (struct ip *)buf;
    iphdrlen = ip->ip_hl << 2;              /*求ip报头长度,即ip报头的长度标志乘4*/
    icmp = (struct icmp *)(buf + iphdrlen); /*越过ip报头,指向ICMP报头*/
    len -= iphdrlen;                        /*ICMP报头及ICMP数据报的总长度*/
    if (len < 8)                            /*小于ICMP报头长度则不合理*/
    {
        printf("ICMP packets\'s length is less than 8\n");
        return -1;
    }
    /*确保所接收的是我所发的的ICMP的回应*/
    if ((icmp->icmp_type == ICMP_ECHOREPLY) && (icmp->icmp_id == pid))
    {
        tvsend = (struct timeval *)icmp->icmp_data;
        tv_sub(&tvrecv, tvsend);                          /*接收和发送的时间差*/
        rtt = tvrecv.tv_sec * 1000 + tvrecv.tv_usec / 1000; /*以毫秒为单位计算rtt*/
        /*显示相关信息*/
        printf("%d byte from %s: icmp_seq=%u ttl=%d rtt=%.3f ms\n",
               len,
               inet_ntoa(from.sin_addr),
               icmp->icmp_seq,
               ip->ip_ttl,
               rtt);
    }
    else
        return -1;
}
```

```
main(int argc, char *argv[])
{
    struct hostent *host;
    struct protoent *protocol;
    unsigned long inaddr = 0l;
    int waittime = MAX_WAIT_TIME;
    int size = 50 * 1024;

    if (argc < 2)
    {
        printf("usage:%s hostname/IP address\n", argv[0]);
        exit(1);
    }

    if ((protocol = getprotobyname("icmp")) == NULL)
    {
        perror("getprotobyname");
        exit(1);
    }
    /*生成使用ICMP的原始套接字,这种套接字只有root才能生成*/
    if ((sockfd = socket(AF_INET, SOCK_RAW, protocol->p_proto)) < 0)
    {
        perror("socket error");
        exit(1);
    }
    /* 回收root权限,设置当前用户权限*/
    setuid(getuid());
    /*扩大套接字接收缓冲区到50K这样做主要为了减小接收缓冲区溢出的
      的可能性,若无意中ping一个广播地址或多播地址,将会引来大量应答*/
    setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size));
    bzero(&dest_addr, sizeof(dest_addr));
    dest_addr.sin_family = AF_INET;

    /*判断是主机名还是ip地址*/
    if (inaddr = inet_addr(argv[1]) == INADDR_NONE)
    {
```

```
        if ((host = gethostbyname(argv[1])) == NULL) /*是主机名*/
        {
            perror("gethostbyname error");
            exit(1);
        }
        memcpy((char *)&dest_addr.sin_addr, host->h_addr, host->h_length);
    }
    else /*是ip地址*/
        memcpy((char *)&dest_addr, (char *)&inaddr, host->h_length);
    /*获取main的进程id,用于设置ICMP的标志符*/
    pid = getpid();
    printf("PING %s(%s): %d bytes data in ICMP packets.\n", argv[1],
           inet_ntoa(dest_addr.sin_addr), datalen);
    send_packet();       /*发送所有ICMP报文*/
    recv_packet();       /*接收所有ICMP报文*/
    statistics(SIGALRM); /*进行统计*/

    return 0;
}
/*两个timeval结构相减*/
void tv_sub(struct timeval *out, struct timeval *in)
{
    if ((out->tv_usec -= in->tv_usec) < 0)
    {
        --out->tv_sec;
        out->tv_usec += 1000000;
    }
    out->tv_sec -= in->tv_sec;
}
```