# 暨南大学本科实验报告专用纸

课程名称 计算机网络实验 成绩评定

实验项目名称 TCP 与 UDP 端口扫描 指导老师 某某某

实验项目编号 9 实验项目类型 综合类 实验地点 N117

学生姓名 某某 学号 xxxxxxxxxx

学院 网络空间安全学院 系 专业 网络空间安全

实验时间 2020 年 12 月 9 日 晚 上～ 2020 年 12 月 9 日 晚 上

## （一）实验目的

1. 了解常用的 TCP、UDP 端口扫描的原理及其各种手段
2. 增强网络安全意识

## （二）实验环境

该实验采用网络结构一

## （三）实验原理

## （四）实验步骤

## （五）实验结果与分析

## （六）附录

基于 Linux 提供的网络编程 API 实现三种端口扫描。为了加快扫描速度，可考虑创建多个线程进行并发扫描（这里指实现简易版）。

首先先随机生成 5 个 50000~60000 的数字作为端口号，绑定套接字，监听对应的端口。以下为 TCP 端口代码如下：

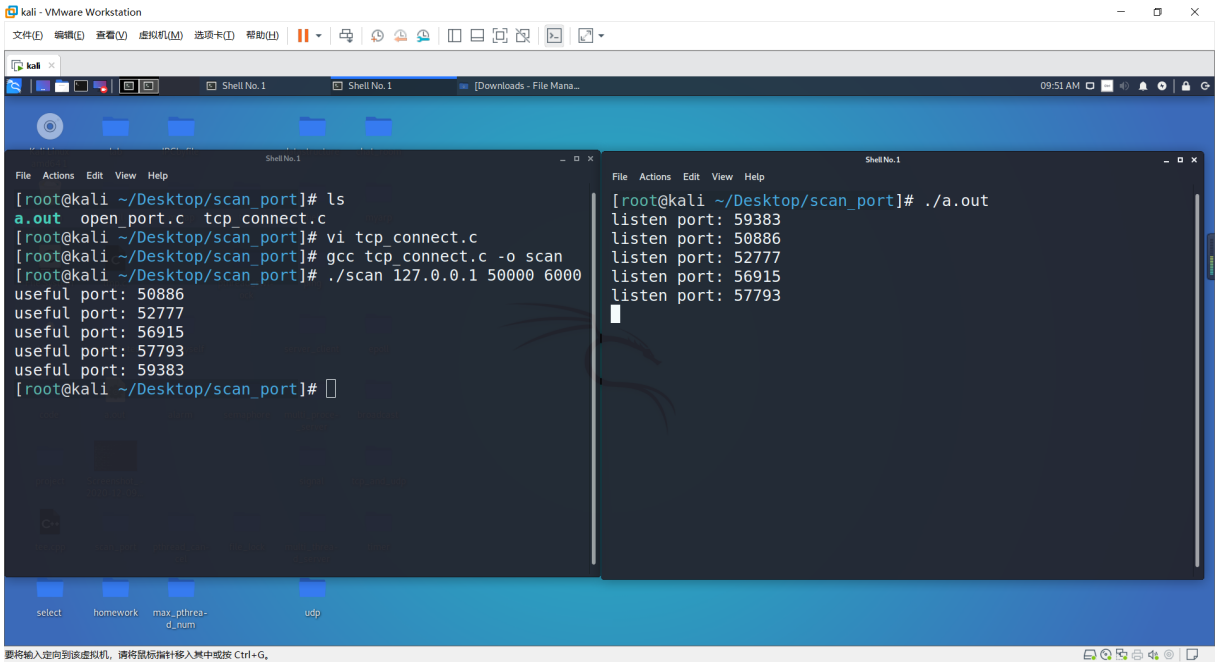
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<arpa/inet.h>
#include<time.h>

int main(void){
    srand(time(NULL));
    int open_ports[5];
    int sockfds[5];
    int flag=1;
    for(int i=0; i<5; ++i){
        /*这里假设生成的随机数不重复(概率很小)*/
        open_ports[i] = rand()%10000 + 50000;
        sockfds[i] = socket(AF_INET,SOCK_STREAM,0);
        struct sockaddr_in servaddr;
        bzero(&servaddr,sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(open_ports[i]);
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

        /*设置端口复用，使得处于time_wait状态的连接地址能重新绑定，便于
            实验*/
        setsockopt(sockfds[i],SOL_SOCKET,SO_REUSEADDR,&flag,sizeof(flag
            ));

        bind(sockfds[i],(struct sockaddr*)&servaddr,sizeof(servaddr));
        listen(sockfds[i],5);
    }
    for(int i=0; i<5; ++i){
        printf("listen port: %d\n",open_ports[i]);
    }
    while(1);
```

```
}
```

TCP Connect 端口扫描

这种扫描方法其实实现非常简单，只需要模拟客户端对目标主机做一次三次连接活动即可，检查 connect 返回值。为了突出整体逻辑，省去了部分检错代码。

```c
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int main(int argc, char **argv)
{
    int sockfd, n;
    struct sockaddr_in servaddr;

    int i;
    for (i = atoi(argv[2]); i < atoi(argv[3]); i++) {
        sockfd = socket(AF_INET, SOCK_STREAM, 0);

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(i);
        inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

        /*每次检查connect函数的返回值即可*/
        if (connect(sockfd, (struct sockaddr*) &servaddr, sizeof(
            servaddr)) < 0) {
            close(sockfd);
        }
        else {
            printf("useful port: %d\n", i);
            close(sockfd);
        }
    }
    return 0;
}
```




TCP SYN 扫描

SYN 扫描就有点难实现了，因为内核封装的接口是实现整个连接，如果只需发送一个同步报文，意味着要自己构建。主要方法包括校验码的计算、报文的发送、报文的接受（因为要判断是不是重置报文）。另外，从**编程的角度**来说这种方式会比较慢（对比 TCP Connect 扫描），因为每次都得重新构造一个 SYN 报文，自行计算检验码，从套接字取得 RST 报文后进行分析，内核态和用户态切换较频繁。

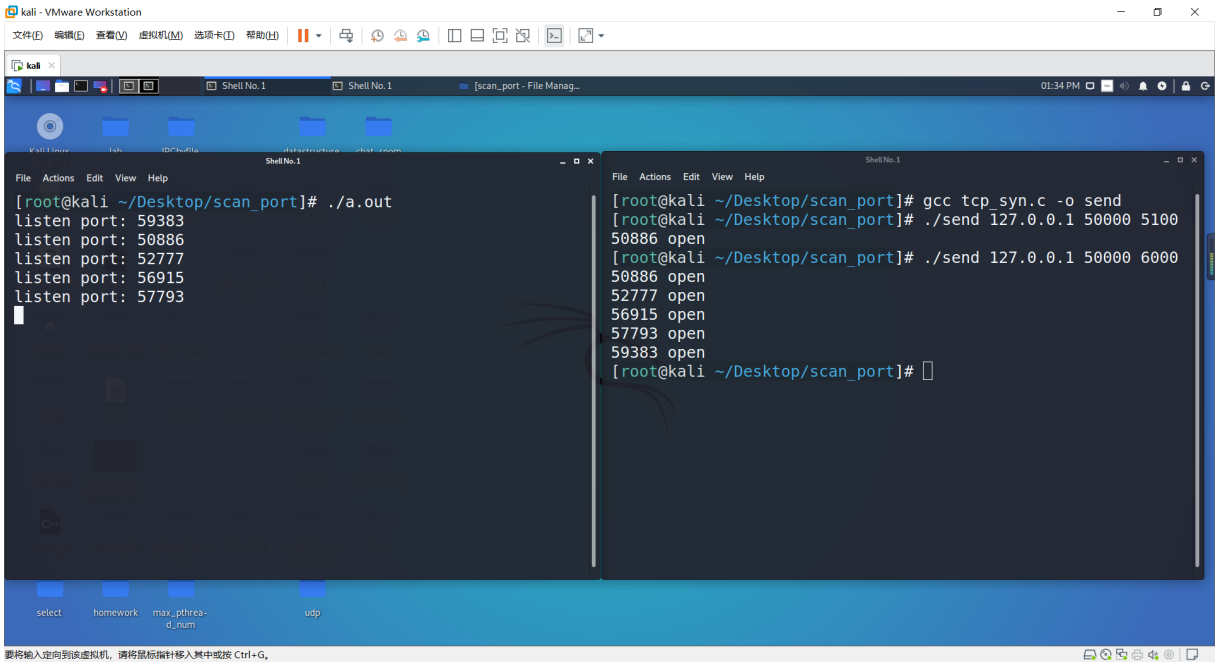
```c
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<linux/tcp.h>
#include<string.h>
#include<netinet/ip.h>
#include<sys/wait.h>
#include<unistd.h>

//计算校验码
unsigned short check_sum(unsigned short *buffer, int size) {
    unsigned long cksum = 0;
    while (size > 1) {
        cksum += *buffer++;
```

```
18          size -= sizeof (unsigned short);
19      }
20      if (size) {
21          cksum += *(unsigned char*) buffer;
22      }
23      cksum = (cksum >> 16) + (cksum & 0xffff);
24      cksum += (cksum >> 16);
25
26      return (unsigned short) (~cksum);
27  }
28
29  void send_data(int sockfd, struct sockaddr_in *addr, int sourceport,
        char sourceip[30]) {
30      char buffer[100];
31      struct iphdr *ip;
32      struct tcphdr *tcp;
33
34      int head_len;
35      int n, i;
36      u_char * pPseudoHead;
37      u_char pseudoHead[12 + sizeof (struct tcphdr) ];
38      u_short tcpHeadLen;
39
40      tcpHeadLen = htons(sizeof (struct tcphdr));
41      head_len = sizeof (struct iphdr) + sizeof (struct tcphdr);
42      bzero(buffer, 100);
43
44      //构建ip头
45      ip = (struct iphdr *) buffer;
46      ip->version = IPVERSION;
47      ip->ihl = sizeof (struct ip) >> 2;
48      ip->tos = 0;
49      ip->tot_len = htons(head_len);
50      ip->id = 0;
51      ip->frag_off = 0;
52      ip->ttl = MAXTTL;
53      ip->protocol = IPPROTO_TCP;
54      ip->check = 0;
55      ip->daddr = addr->sin_addr.s_addr;
56      ip->saddr = inet_addr(sourceip);
57
```

```
58
59      //构建TCP头
60      tcp = (struct tcphdr *) (buffer + sizeof (struct ip));
61      tcp->source = htons(sourceport);
62      tcp->dest = addr->sin_port;
63      tcp->seq = htonl(30000);
64      tcp->ack_seq = 0;
65      tcp->doff = 5;
66      tcp->syn = 1;
67      tcp->urg_ptr = 0;
68      tcp->window = htons(10052);
69
70
71      //构建伪首部
72      pPseudoHead = pseudoHead;
73      memset(pPseudoHead, 0, 12 + sizeof (struct tcphdr));
74      memcpy(pPseudoHead, &ip->saddr, 4);
75      pPseudoHead += 4;
76      memcpy(pPseudoHead, &ip->daddr, 4);
77      pPseudoHead += 4;
78      memset(pPseudoHead, 0, 1);
79      pPseudoHead++;
80      memset(pPseudoHead, 0x0006, 1);
81      pPseudoHead++;
82      memcpy(pPseudoHead, &tcpHeadLen, 2);
83      pPseudoHead += 2;
84      memcpy(pPseudoHead, tcp, sizeof (struct tcphdr));
85
86      tcp->check = 0;
87      tcp->check = check_sum((unsigned short *) pseudoHead, sizeof (
            struct tcphdr) + 12);
88      if (sendto(sockfd, buffer, head_len, 0, (struct sockaddr *) addr, (
            socklen_t)sizeof (struct sockaddr_in)) < 0) {
89          perror("sendto");
90      }
91  }
92
93  void  recv_packet(const char* localIP, int localPort, int sockfd, int
        startport, int endport) {
94      struct tcphdr * tcp;
95      char *srcaddr;
```

```
96      int loopend;
97      int size;
98      char readbuff[1600];
99      struct sockaddr_in from;
100     int from_len,n;
101
102     tcp = (struct tcphdr *) (readbuff + 20); /*那个sockfd中读出的数据包
            括了IP头的所以+20*/
103     for (n = startport; n < endport + 1; n++) {
104         size = recv(sockfd, readbuff, 1600, MSG_DONTWAIT);
105         if (size < (20 + 20))/*读出的数据小于两个头的最小长度的话
                continue*/
106         continue;
107         if (ntohs(tcp->dest) != localPort)
108         continue;
109         if (tcp->rst && tcp->ack)/*端口关闭或者没有服务*/
110         continue;
111         if (tcp->ack && tcp->syn)/*端口开启*/ {
112             printf("%5u open\n", (ntohs(tcp->source)));
113             fflush(stdout);
114             continue;
115         }
116     }
117 }
118 int main(int argc, char** argv){
119     if(argc!=4){
120         return -1;
121     }
122     char ip[30];
123     int myport = 8888;
124     char* myip = "192.168.145.128";
125     int flag = 1;
126     strcpy(ip,argv[1]);
127     int startport = atoi(argv[2]);
128     int endport = atoi(argv[3]);
129     struct sockaddr_in addr;
130     bzero(&addr,sizeof(addr));
131     addr.sin_family = AF_INET;
132     inet_pton(AF_INET,ip,&addr.sin_addr);
133     int sockfd = socket(AF_INET,SOCK_RAW,IPPROTO_TCP);
134     setsockopt(sockfd,IPPROTO_IP,IP_HDRINCL,&flag,sizeof(flag));
```

```
    for(int i=startport; i<=endport; ++i){
        addr.sin_port = htons(i);
        send_data(sockfd,&addr,myport,myip);
        recv_packet(myip,myport,sockfd,startport,endport);
    }
    return 0;
}
```



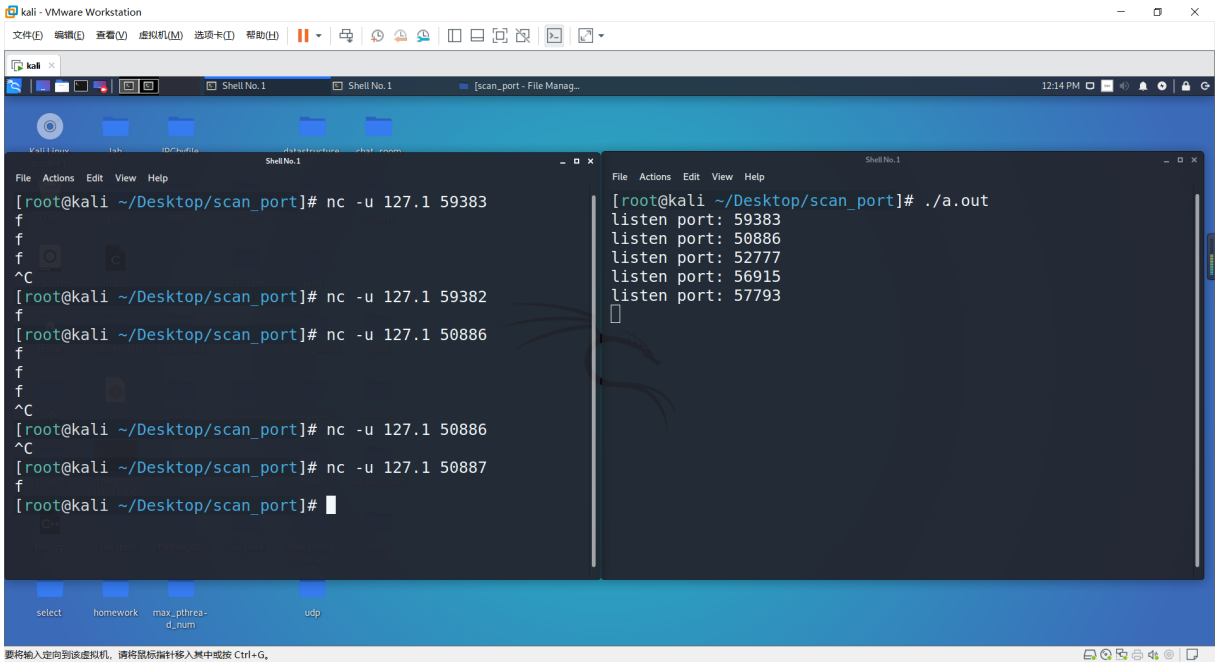

TCP UDP 扫描

给一个未开放 UDP 协议的端口写数据会收到 ICMP 错误报文，我试了一下资料里介绍的，使用 recvfrom 函数、sendto 函数、write 函数进行测试都没有成功，关掉防火墙后也依然没有成功。

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<errno.h>
#include<fcntl.h>

const int myport = 8888;
const char* myip = "192.168.145.128";

```

```
15 int main(int argc, char** argv){
16     if(argc != 4){
17         return -1;
18     }
19     int flag = 1;
20     char dstip[30];
21     strcpy(dstip,argv[1]);
22     int startport = atoi(argv[2]);
23     int endport = atoi(argv[3]);
24 
25     int sockfd = socket(AF_INET,SOCK_DGRAM,0);
26     /*
27     int fl = fcntl(sockfd,F_GETFL,0);
28     fcntl(sockfd,F_SETFL,fl|O_NONBLOCK);
29     */
30     struct sockaddr_in servaddr, cliaddr;
31     bzero(&servaddr,sizeof(servaddr));
32     bzero(&cliaddr,sizeof(cliaddr));
33 
34     servaddr.sin_family = AF_INET;
35     inet_pton(AF_INET,dstip,&servaddr.sin_addr);
36     const char* string = "hello";
37     char buf[BUFSIZ];
38     for(int i=startport; i<=endport; ++i){
39         servaddr.sin_port = htons(i);
40         flag = 1;
41         for(int j=0 ;j<3; ++j){
42             //int ret = sendto(sockfd,string,strlen(string),0,(struct
                sockaddr*)&servaddr,sizeof(servaddr));
43             int ret = write(sockfd,string,strlen(string));
44             if(ret<0){
45                 flag = 0;
46                 break;
47             }
48         }
49         if(flag){
50             printf("open port: %d\n",i);
51         }
52         /*
53         int len = sizeof(servaddr);
54         int ret = recvfrom(sockfd,buf,BUFSIZ,MSG_DONTWAIT,(struct
```

```
                sockaddr*)&servaddr,&len);
55          if(errno!=EAGAIN){
56              printf("open port: %d\n",i);
57          }
58          */
59      }
60      return 0;
61  }
```




不过，在命令行界面访问未开放的 UDP 端口，可见写入数据时会自动返回，猜想应该是系统收到了 ICMP 错误报文给进程发送了终止信号。所以有一点可以肯定，向未开放的 UDP 端口发送数据一定会收到错误报文，但使用系统调用，诸如 write、sendto、recvfrom 等并不一定能检测。