南开大学

Nankai University

南　　开　　大　　学

网 安 学 院

网络安全技术实验报告

第四次作业

沙璇 1911562

年级：2019 级

专业：信息安全

提交日期：2022/6/11

2022 年 6 月 12 日

摘要

端口扫描器的设计与实现

关键字：端口扫描, linux

目录

一、 实验目的

端口扫描器是一种重要的网络安全检测工具。通过端口扫描，不仅可以发现目标主机的开放端口和操作系统的类型，还可以查找系统的安全漏洞，获得弱口令等相关信息。因此，端口扫描技术是网络安全的基本技术之一，对于维护系统的安全性有着十分重要的意义。

本章编程训练的目的如下：

（1）掌握端口扫描器的基本设计方法。

（2）理解 ping 程序，TCP connect 扫描，TCP SYN 扫描，TCP FIN 扫描以及 UDP 扫描的工作原理。

（3）熟练掌握 Linux 环境下的套接字编程技术。

（4）掌握 Linux 环境下多线程编程的基本方法。

二、 实验要求

本章编程训练的要求如下：

（1）编写端口扫描程序，提供 TCP connect 扫描，TCP SYN 扫描，TCP FIN 扫描以及 UDP 扫描 4 种基本扫描方式。

（2）设计并实现 ping 程序，探测目标主机是否可达

三、 实验内容

本次实验使用 python 编写，在 kali 中运行。

涉及到的主要源文件有：ping.py、main.py、tcp-conn.py、tcp-syn.py、tcp-fin.py、udp.py。完整代码见附件。

1.ping.py：ping 程序的实现。

2.main.py：该文件是程序的主文件，用于完成端口扫描程序中功能的选择和调用。

3.tcp-conn.py：connect 扫描的实现。

4.tcp-syn.py：syn 扫描的实现。

5.tcp-fin.py：fin 扫描的实现。

(一) ping 程序

ping 程序是日常网络管理中经常使用的程序。它用于确定本地主机与网络中其它主机的通信情况。因为只是简单地探测某一 IP 地址所对应的主机是否存在，因此它的原理十分简单。扫描发起主机向目标主机发送一个要求回显（type = 8）的 ICMP 数据包，目标主机在收到请求后，会返回一个回显（type = 0）的 ICMP 数据包。扫描发起主机可以通过是否接收到响应的 ICMP 数据包来判断目标主机是否存在。

在本章编程中，可以在向目标主机发起端口扫描之前使用 ping 程序确定目标主机是否存在。如果 ping 目标主机成功，则继续后面的扫描工作；否则，放弃对目标主机的扫描。

(二) 常用扫描技术

1. TCP 扫描

TCP 协议的标志位对于扫描来说至关重要；TCP 数据报头标志位示意图如图1所示

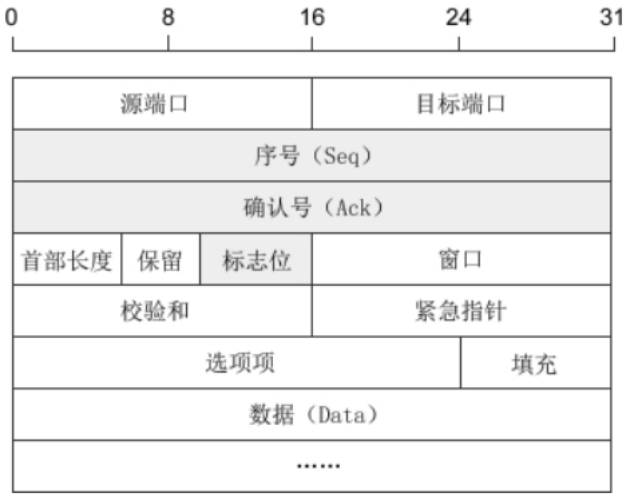

图 1: TCP 数据报头标志位

各个标志位的含义如下：

SYN：标志位用来建立连接，让连接双方同步序列号。如果 SYN = 1 而 ACK=0，则表示该数据包为连接请求，如果 SYN=1 而 ACK=1 则表示接受连接。

FIN：表示发送端已经没有数据要求传输了，希望释放连接。

RST：用来复位一个连接。ST 标志置位的数据包称为复位包。一般情况下，如果 CP 收到的一个分段明显不是属于该主机上的任何一个连接，则向远端发送一个复位包。

URG：紧急数据标志。如果为 1，表示本数据包中包含紧急数据。此时紧急数据指针有效。

ACK：为确认标志位。如果为 1，表示包中的确认号时有效的。否则，包中的确认号无效。

PSH：如果置位，接收端应尽快把数据传送给应用层。而一个 TCP 的连接过程，描述如下：

1) 首先客户端 (请求方) 在连接的请求中，发送 SYN=1，ACK=0 的 TCP 数据包给服务器 (接受请求端)，表示要求同服务器端建立一个请求。

2) 如果服务器端 (接受端) 响应这个请求，就返回一个 SYN=1 且 ACK=1 的数据包给客户端，表示服务器统一这个连接，并要求客户端进行确认；

3) 最后客户端发送 SYN=0,ACK=1 的数据包给服务器端，表示确认连接。

在本次实验中主要使用的 TCP 扫描包含了以下的几种形式：

TCP Connect **扫描** Connect 扫描的原理非常简单，由扫描主机调用系统的 API Connect 尝试连接目的主机指定端口，如果 connect 成功，意味着扫描主机与被扫描主机之间发生了一次完整的 TCP 三次握手建立过程，表示该端口开放，否则代表端口关闭。但是 connect 扫描的效率低下，由于 TCP 协议是可靠的协议，connect 系统调用不会在尝试发送第一个数据包未得到响应就放弃，而是会经过多次尝试后才彻底的放弃，所需要的时间较长，此外 connect 失败会在系统中造成大量的失败日志，容易被系统管理员发现。

TCP SYN **扫描** TCP SYN 扫描是使用最为广泛的扫描方式，其原理就是向待扫描的端口发送 SYN 数据包，如果能够收到 SYN+ACK 数据包就代表此端口是开放的，如果收到 RST 数据包，则证明此端口关闭，如未收到任何数据包且确定该主机是存在的，则证明该端口是被防火墙过滤了，由于 SYN 扫描并不会完成 TCP 三次握手过程，所以 SYN 扫描又叫做半开放扫描。SYN 扫描最大的优点就是速度快，在 Internrt 上如果不存在防火墙，SYN 扫描每秒钟可以扫描数千个端口。但是 SYN 扫描由于其扫描的行为较为明显，容易被入侵监测系统发现，也容易被防火墙屏蔽，且构造原始数据包需要有较高的系统权限。

TCP FIN **扫描** FIN 扫描是对某个 IP 地址特定端口发送一个 TCP FIN 数据包给远端主机。如果主机没有任何反馈，且能够确定这个主机是存在的，则主机正在监听这个端口；主机反馈 TCP RST，说明主机是存在的，但是没有监听这个端口；FIN 扫描具有较好地隐瞒性，不会留下日志，但是应用具有很大的局限性：由于不同的系统实现网络协议栈的细节不同，FIN 扫描只能扫描 Linux/UNIX 系统，如果是 Windows 系统，无论端口是否开放都会直接返回 RST 数据包，无法对端口的状态进行判断。

2. UDP **扫描**

一般情况下，当向一个关闭的 UDP 端口发送数据时，目标主机会返回一个 ICMP 不可达（ICMP port unreachable）的错误。UDP 扫描就是利用了上述原理，向被扫描端口发送 0 字节的 UDP 数据包，如果收到一个 ICMP 不可达响应，那么就认为端口是关闭的；而对于那些长时间没有响应的端口，则认为是开放的。

但是，因为大部分系统都限制了 ICMP 差错报文的产生速度，所以针对特定主机的大范围 UDP 端口扫描的速度非常缓慢。此外，UDP 协议和 ICMP 协议是不可靠协议，没有收到响应的情况也可能是由于数据包丢失造成的，因此扫描程序必须对同一端口进行多次尝试后才能得出正确的结论。UDP 数据报如图2所示。

图 2: MD5

UDP 数据报格式有首部和数据两个部分。首部很简单，共 8 字节。包括：

源端口（Source Port）：2 字节，源端口号。

目的端口（Destination Port ）：2 字节，目的端口号。

长度（Length）：2 字节，UDP 用户数据报的总长度，以字节为单位。

检验和（Checksum）：2 字节，用于校验 UDP 数据报的数字段和包含 UDP 数据报首部的“伪首部”。其校验方法同 IP 分组首部中的首部校验和。

伪首部，又称为伪包头（Pseudo Header）：是指在 TCP 的分段或 UDP 的数据报格式中，在数据报首部前面增加源 IP 地址、目的 IP 地址、IP 分组的协议字段、TCP 或 UDP 数据报的总长度等共 12 字节，所构成的扩展首部结构。此伪首部是一个临时的结构，它既不向上也不向下传递，仅仅只是为了保证可以校验套接字的正确性。

(三) 具体代码

MD5 程序分为 6 个部分。1.ping.py：ping 程序的实现。

2.main.py：该文件是程序的主文件，用于完成端口扫描程序中功能的选择和调用。

3.tcp-conn.py：connect 扫描的实现。
4.tcp-syn.py：syn 扫描的实现。
5.tcp-fin.py：fin 扫描的实现。

1. ping **程序**

ping 命令主要基于 ICMP 实现，它包含了两部分：客户端、服务器。
客户端：向服务端发送 ICMP 回显请求报文
服务端：向客户端返回 ICMP 回西显响应报文
ICMP 报文通用格式如下：
类型：1 个字节。8 表示回显请求报文，0 表示回显响应报文。
代码：1 个字节。回显请求报文、回显响应报文时均为 0。
校验和：2 个字节。非重点，略过。
标识符：2 个字节。发送 ICMP 报文的客户端进程的 id，服务端会回传给客户端。
序列号：2 个字节。从 0 开始，客户端每次发送新的回显请求时 +1。服务端原样会传。
数据：6 个字节。客户端记录回显请求的发送时间，服务端记录回西显响应的发送时间
此处仅截取主要函数 ping 如下

ping

```
1
2
3  def ping(host):
4      send, accept, lost = 0, 0, 0
5      sumtime, shorttime, longtime, avgtime = 0, 1000, 0, 0
6      #TODO icmp数据包的构建
7      data_type = 8 # ICMP Echo Request
8      data_code = 0 # must be zero
9      data_checksum = 0 # "...with value 0 substituted for this field..."
10     data_ID = 0 #Identifier
11     data_Sequence = 1 #Sequence number
12     payload_body = b'abcdefghijklmnopqrstuvwabcdefghi' #data
13
14     # 将主机名转ipv4地址格式，返回以ipv4地址格式的字符串，如果主机名称是ipv4
           地址，则它将保持不变
15     dst_addr = socket.gethostbyname(host)
16     print("正在 Ping {0} [{1}] 具有 32 字节的数据:".format(host,dst_addr))
17     for i in range(0,4):
18         send = i + 1
19         #请求ping数据包的二进制转换
20         icmp_packet = request_ping(data_type,data_code,data_checksum,data_ID,
               data_Sequence + i,payload_body)
21         #连接套接字,并将数据发送到套接字
22         send_request_ping_time,rawsocket,addr = raw_socket(dst_addr,
               icmp_packet)
23         #数据包传输时间
24         times = reply_ping(send_request_ping_time,rawsocket,data_Sequence + i
               )
```

```
        if times > 0:
            print("来自 {0} 的回复: 字节=32 时间={1}ms".format(addr,int(times
                *1000)))

            accept += 1
            return_time = int(times * 1000)
            sumtime += return_time
            if return_time > longtime:
                longtime = return_time
            if return_time < shorttime:
                shorttime = return_time
            time.sleep(0.7)
        else:
            lost += 1
            print("请求超时。")

        if send == 4:
            print("{0}的Ping统计信息:".format(dst_addr))
            print("\t数据包: 已发送={0},接收={1}, 丢失={2} ({3}%丢失) , \n往
                返行程的估计时间 (以毫秒为单位) : \n\t最短={4}ms, 最长={5}ms
                , 平均={6}ms".format(
                i + 1, accept, i + 1 - accept, (i + 1 - accept) / (i + 1) *
                    100, shorttime, longtime, sumtime/send))
```

2. main.py

本文件主要用于功能选择。

main.py

```
from TCP_CONN import conn_scanner
from TCP_SYN import syn_scanner
from TCP_FIN import fin_scanner
from UDP import udp_scanner

def menu():
    print('''
     ----菜单

     --------1. TCP_CONN扫描
     --------2. TCP_SYN扫描
     --------3. TCP_FIN扫描
     --------4. UDP扫描
     --------5. 显示菜单
     --------6. 退出
     ''')

def main():
    targetIP=input("请输入目标IP: ")
```

```
    a=int(input("请输入扫描起始端口: "))
    b=int(input("请输入扫描终止端口: "))
    portslist=(list(range(a,b)))
    menu()
    while True:
        try:
            options=int(input("请输入扫描方式: "))
        except:
            continue
        if options==1:
           conn_scanner(targetIP,portslist)
        elif options==2:
           syn_scanner(targetIP,portslist)
        elif options==3:
           fin_scanner(targetIP,portslist)
        elif options==4:
           udp_scanner(targetIP,portslist)
        elif options==5:
            menu()
        elif options==6:
            break
        else:
            continue
main()
```

3. tcp-conn.py

端口扫描的特征码:

Connect 扫描（端口开放):

1、客户端发送包: URG=0, ACK=0, PSH=0, RST=0, SYN=1, FIN=0

2、服务端回包: URG=0, ACK=1, PSH=0, RST=0, SYN=1, FIN=0

3、客户端发送包: URG=0, ACK=1, PSH=0, RST=0, SYN=0, FIN=0

4、客户端发送包: URG=0, ACK=1, PSH=0, RST=1, SYN=0, FIN=0

输入起始端口与终止端口，判断是否为开放端口。

tcp-conn

```

import socket
import time
import threading

targetIP="127.0.0.1"

a=int(input("请输入扫描起始端口: "))
b=int(input("请输入扫描终止端口: "))
portslist=(list(range(a,b)))
'''
```

```
portslist=[21,22,23,80,135,139,445]
'''
def conn_scan(ip,port):
    scansocket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    try:
        status=scansocket.connect_ex((ip,port))
        if status==0:
            print(f"[+] Port {str(port)} Is Open\n")
    except:
        print("error")
    scansocket.close()
def conn_scanner(targetIP,portslist):
    print(f"Scanning {targetIP} for Open TCP_CONN Ports\n")
    for i in portslist:
        run=threading.Thread(target=conn_scan,args=(targetIP,i))
        run.start()
        run.join()

conn_scanner(targetIP,portslist)
```

4. tcp-syn.py

端口扫描的特征码：
SYN 扫描（端口开放）：
1、客户端发送包：URG=0，ACK=0，PSH=0，RST=0，SYN=1，FIN=0
2、服务端回包：URG=0，ACK=1，PSH=0，RST=0，SYN=1，FIN=0
3、客户端发送包：URG=0，ACK=0，PSH=0，RST=1，SYN=0，FIN=0
端口未开放时：（Connect 和 SYN 扫描数据包一样）
1、客户端发送包：URG=0，ACK=0，PSH=0，RST=0，SYN=1，FIN=0
2、服务端回包：URG=0，ACK=1，PSH=0，RST=1，SYN=0，FIN=0
输入起始端口与终止端口，判断是否为开放端口或未开放端口。

tcp-syn

```

import logging
import threading
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.layers.inet import IP, TCP, UDP, ICMP
from scapy.all import *

#target = str(input("请输入目标IP: "))
target="127.0.0.1"
a=int(input("请输入扫描起始端口: "))
b=int(input("请输入扫描终止端口: "))
portslist=(list(range(a,b)))
'''
portslist=[21, 22, 34, 135, 139, 80, 445]
```

```
'''
def syn_scan(port):
    sport = RandShort()
    pkt = sr1(IP(dst=target) / TCP(sport=sport, dport=port, flags="S"),
        timeout=1, verbose=0)
    if pkt != None:
        if pkt.haslayer(TCP):
            if pkt[TCP].flags == 18:
                print(f"[+] Port {str(port)} Is Open\n")
    else:
        print(f"[+] Port {str(port)} Is Close\n")

def syn_scanner(target,portslist):
    print(f"Scanning {target} for Open TCP_SYN Ports\n")
    for x in portslist:
        threading.Thread(target=syn_scan,args=(x,)).start()

#syn_scanner(target,portslist)
#print('Scan Is Completed!\n')
```

5. tcp-fin.py

FIN 扫描（端口未开放）：
1、客户端发送包：URG=0，ACK=0，PSH=0，RST=0，SYN=0，FIN=1
2、服务端回包：URG=0，ACK=1，PSH=0，RST=1，SYN=0，FIN=0
输入起始端口与终止端口，判断是否为开放端口或未开放端口。

tcp-fin

```

from scapy.layers.inet import IP, TCP
from scapy.sendrecv import sr, sr1
import threading

'''
适用于Linux设备
通过设置flags位为'FIN',不回复则表示端口开启，回复并且回复的标志位为RST表示端口关闭
'''
targetIP="127.0.0.1"
a=int(input("请输入扫描起始端口："))
b=int(input("请输入扫描终止端口："))
portslist=(list(range(a,b)))
'''
portslist=[21,22,23,80,135,139,445]
'''
def fin_scan(targetIP,port):
    p = IP(dst=targetIP) / TCP(dport=int(port), flags="F")
    ans = sr1(p, timeout=1, verbose=0)
```

```
    if sr1(p, timeout=1, verbose=0) == None:
        print(f"[+] Port {str(port)} Is Open\n")
    elif ans != None and ans[TCP].flags == 'RA':
        #ans.display()
        #print(f"[+] Port {str(port)} Is Close\n")
        pass

def fin_scanner(targetIP,portslist):
    print(f"Scanning {targetIP} for Open TCP_FIN Ports\n")
    for p in portslist:
        threading.Thread(target=fin_scan,args=(targetIP,p)).start()

#scanner(targetIP,portslist)
```

6. udp

udp

```

from scapy.all import *
from scapy.layers.inet import IP, UDP
import threading

target="127.0.0.1"
a=int(input("请输入扫描起始端口："))
b=int(input("请输入扫描终止端口："))
portslist=(list(range(a,b)))
'''
portslist=[21, 22, 34, 135, 139, 80, 445]
'''
def UDP_scan(target,port):
    pkt=IP(dst=target)/UDP(dport=int(port))
    res=sr1(pkt,timeout=0.1,verbose=0)
    if res==None:
        print(f"[+] Port {str(port)} Is Open\n")

def udp_scanner(target,portslist):
    print(f"Scanning {target} for Open UDP Ports\n")
    for port in portslist:
        t=threading.Thread(target=UDP_scan,args=(target,port))
        t.start()

if __name__=='__main__':
    udp_scanner(target,portslist)
```

四、 实验结果

在 kali 系统中本文件夹下运行 sudo python3 ping.py ping 结果如图3所示

图 3: ping

无丢失，ping 通。

输入 sudo python3 main.py 进入功能选择界面，如图4所示

图 4: main

选择 tcp-connect 模式，端口范围为 80 140

结果如图5所示

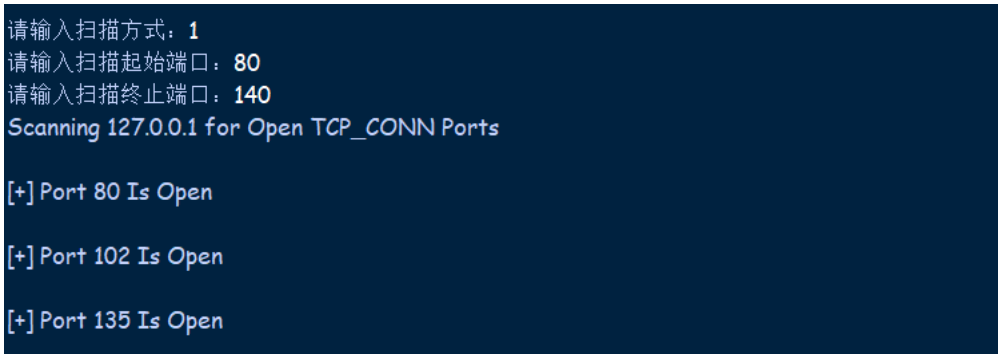

图 5: conn 校验

选择 tcp-syn 模式，端口范围为 80 140

结果如图6所示

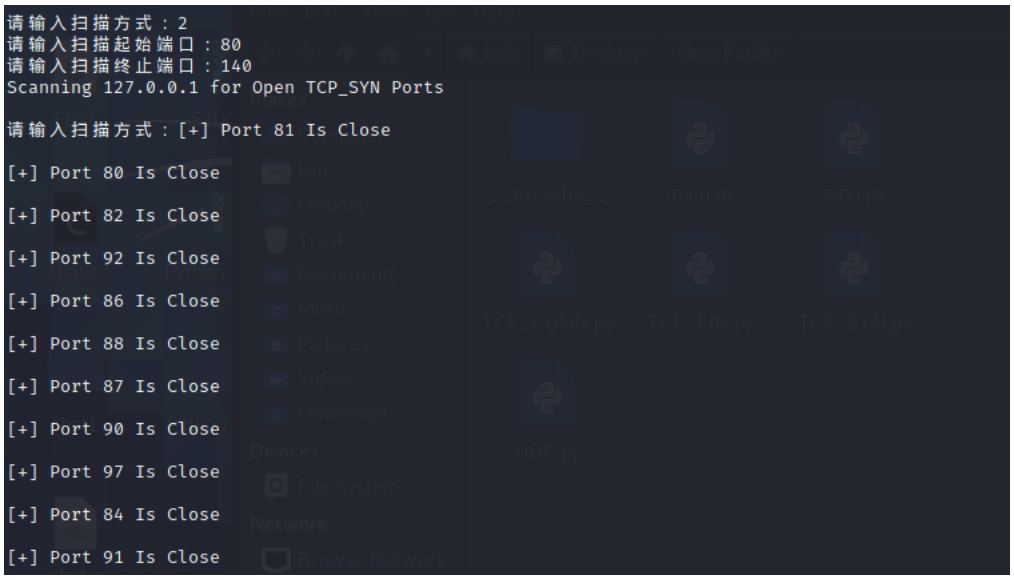

图 6: syn

选择 tcp-fin 模式，端口范围为 80 140
结果如图7所示

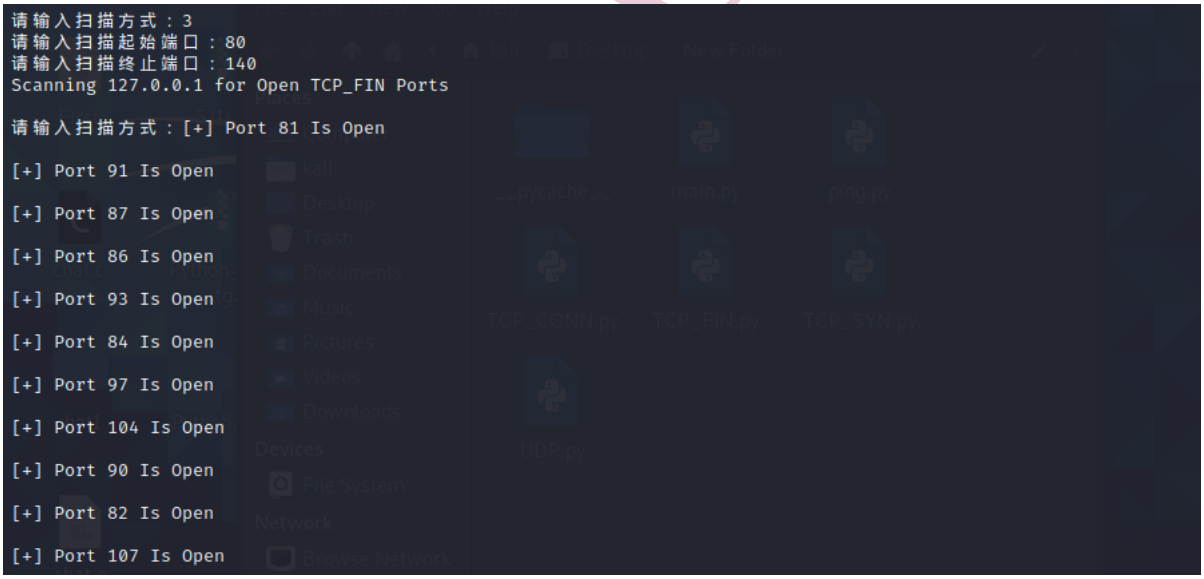

图 7: fin

选择 udp 模式，端口范围为 80 140
结果如图8所示

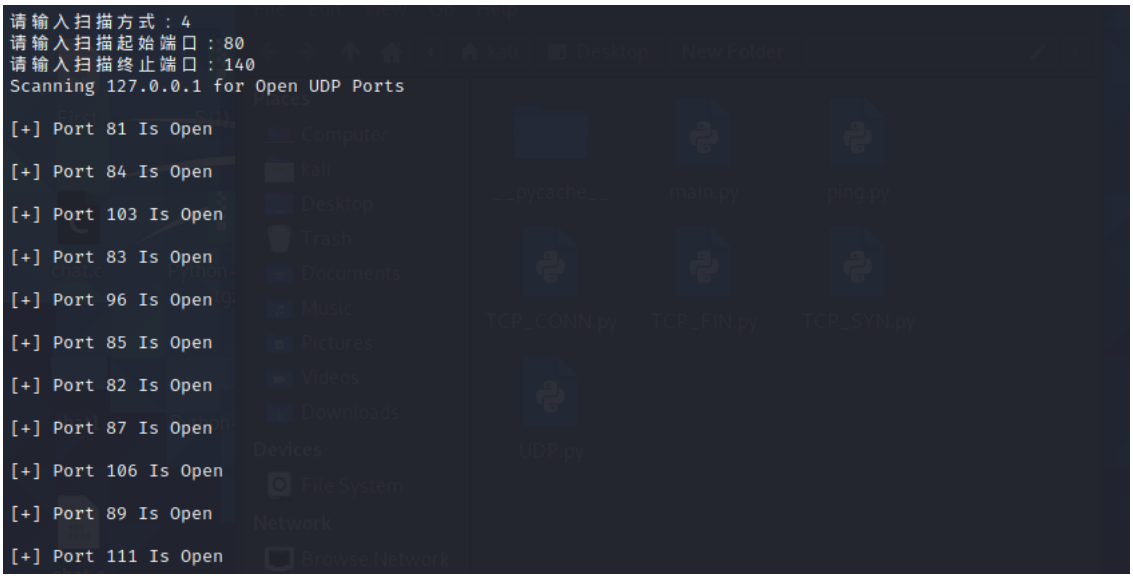

图 8: udp

五、 总结

通过这次的课程设计，对网络编程有了更深入的了解，进一步熟悉了 TCP 和 UDP 协议的内容，掌握了 TCP、UDP 扫描端口的基本原理。对编程思想有了进一步的体会，养成了一些良好的编程习惯。