

## netcat 简易版

**摘要：**netcat 被称为网络界的瑞士军刀。为了更好地学习计算机网络，同时满足个人兴趣，在本次报告中介绍了一个简易版 netcat 的设计、实现和测试。netcat 简易版包含 3 个主要功能：命令执行、文件上传、流量嗅探。本次工具的通过本地机和虚拟机进行测试。

**关键字：**netcat; tcp; socket。

**代码行数：**400 行

### 1. 项目背景和意义

netcat 是一个计算机网络公用程序，用来对网络连线 TCP 或者 UDP 进行读写。一般来说，一台 Linux 主机自带 netcat，但在实际生产环境中，由于其具有一定的危险性，厂商会去除服务器上的 netcat。当我们通过一些漏洞进入一台服务器时，没有 netcat 但是安装了 Python，我们就可以编写一个类 netcat 应用用于持久性的连接。

最近在研究计算机网络，出于兴趣和学习，编写本次脚本，以达到学习的目的。

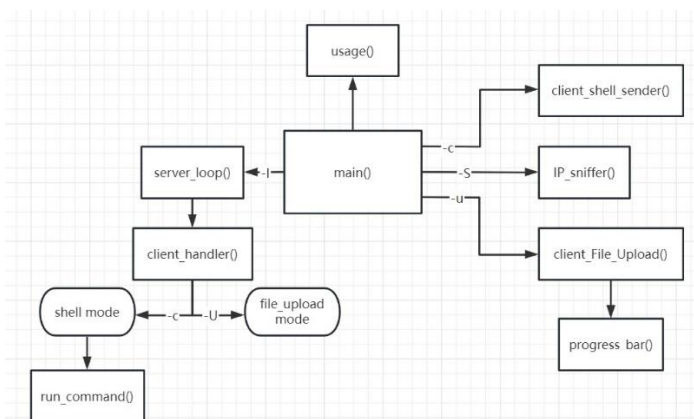
### 2. 需求分析

为了满足我们的目的，首先应该具有文件上传功能，然后就是命令执行。文件上传和命令执行两个功能的代码应该完全分离，因为在实际利用时会先编写简单的文件上传，再通过其将完整代码上传。本次额外添加了目标主机的流量监测功能。

本次基于命令行实现，对命令行进行了一些美化，3xsh0re 为常用网名。

### 3. 概要和详细设计

#### 3.1 整体设计



用到的 Python 库: ctypes, ipaddress, sys, os, socket, struct, getopt, threading, subprocess, colorama

代码主要分为客户端和服务端两个部分,除了用于美化命令行的 colorama 库和用于定义 IP 类的 ctypes 库,其他库均为 Python 自带的库。整个程序假设我们在目标服务器上已经拥有开启端口的权限,这里虚拟机采用 root 账户登陆。

### 3.2 服务端设计

定义了一个主函数 server\_loop,其开启了一个套接字,默认监听 127.0.0.1,最多监听 10 个线程,然后循环接收客户端连接,每有一个新的客户端进入,开启一个新的线程,并调用处理函数 client\_handler。

定义了一个客户端线程处理函数 client\_handler,分为两大块,一块处理文件上传,一块处理命令执行。下面依次介绍两部分代码:

```
if file:
    print(Fore.YELLOW + "[*]FileUp
mode\n~~~~~" + Fore.RESET)
    flag = ""
    # 接受保存文件名
    while "BEGIN_RECEIVE" not in flag:
        flag+= client_socket.recv(1).decode()
    server_file_save = flag.replace("BEGIN_RECEIVE","")
    print(Fore.GREEN + "[*]Begin Receive" + Fore.RESET)
    # 接收文件字节流
    file_buffer = ""
    while True:
        data = client_socket.recv(1024)
        if data==b'FILE_UPLOAD_OVER':
            print(Fore.GREEN + "[*]File receive over!" +
Fore.RESET)
            break
        else:
            file_buffer += data.decode()
            pass
    # 接收文件流并写入
    try:
        print(Fore.GREEN + f"[*]client
{addr[0]}:{addr[1]} file save path is ./ {server_file_save}" +
Fore.RESET)
        file_descriptor = open("./" + server_file_save,
"w")
        file_descriptor.write(file_buffer)
        file_descriptor.close()
        # 确认文件写入
        client_socket.send(f"[*]Successfully saved file
to {os.getcwd()}/{server_file_save}\n".encode())
    except:
        client_socket.send(f"[*]Failed to save file to
{server_file_save}\n".encode())
```

文件上传的处理，首先循环接收"BEGIN\_RECEIVE"标志，表示文件名接收完毕，开始接收文件内容，`file_buffer` 变量用于记录文件字符，`data` 变量用于接收 2 进制文件流，当接收到'FILE\_UPLOAD\_OVER'标志时，表示文件内容已经写完，开始写入文件，成功与否会反馈给客户端。

```
if command:
    print(Fore.YELLOW + "\n[*]Shell
mode\n~~~~~" + Fore.RESET)
    try:
        client_socket.send("Get Shell!\n".encode())
        while True:
            # 进入命令行
            cmd_buffer = ""
            while "\n" not in cmd_buffer:
                cmd_buffer +=
client_socket.recv(1024).decode()
                print(Fore.GREEN + f"[*]command:
{cmd_buffer}" + Fore.RESET,end='')
            if (cmd_buffer=="user_exit\n"):
                print(Fore.GREEN + "[*]a client exit!" +
Fore.RESET)
                break
            # 服务端的返回
            resp = run_command(cmd_buffer,client_os)
            # 处理无回显的命令
            if resp == "":
                client_socket.send("\n".encode())
            else:
                client_socket.send(resp.encode())
            pass
        except:
            print(Fore.RED + "[*]something went wrong when
execute shell!" + Fore.RESET)
```

命令执行的处理，首先向客户端发送"Get Shell!"标识表示服务端已经开启命令执行的处理，等待客户端的输入，定义一个 cmd\_buffer 变量用于接收用户输入的命令，一直接收知道接收到"\n"为止，然后调用 run\_code 函数处理命令输入，如果是无回显的命令，返回一个"\n"，否则返回回显内容。当接收到"user\_exit\n"命令时，打印一个客户断开连接的显示。

```
def run_command(cd,client_os):
    cd = cd.rstrip()
    cd_list=cd.split()
    # 单独处理 ping 命令
    if "ping" in cd_list:
        ip_address=cd_list[1]
        # 开启一个子进程处理 ping 命令
        ping_process=subprocess.Popen(["ping","-c", "4",ip_address],stdout=subprocess.PIPE)
        output,error1=ping_process.communicate()
        return output.decode()
    # 单独处理 windows 下的清空
    if cd == "clear" and client_os=="nt":
        return "cls\n"
    # 处理命令输入
    try:
        output = subprocess.check_output(cd,
        stderr=subprocess.STDOUT, shell=True).decode()
        pass
    except:
        output = "Failed to execute command.\r\nMaybe wrong command?\r\n"
    return output
```

命令执行函数，主要使用了 Python 的 subprocess 库执行命令，对 ping 命令做了单独的处理，防止 ping 命令的进程无法终止导致客户端失去连接，对 windows 的清屏函数做了单独的处理，防止 Windows 不能识别 Linux 主机返回的清屏指令。

### 3.3 客户端设计

客户端同样分为两部分，文件上传和命令执行，定义了两个函数 client\_File\_Upload 和 client\_shell\_sender，下面同样依次解读两部分的代码。

```
def client_File_Upload():
    global client_upload
    global server_file_save
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client.connect((target,port))
        print(Fore.GREEN + f"[*]connect to {target}:{port}
successfully!" + Fore.RESET)
        # 打开文件并读取
        with open(client_upload,"rb") as file:
            i = 0
            # 发送保存文件名
            while i<=1:
                time.sleep(0.1)
                client.send((server_file_save+" BEGIN_RECEIVE
").encode())
                i+=1
                pass
            total_bytes = os.path.getsize(client_upload)
            bytes_sent = 0
            for b_txt in file:
                # 发送二进制数据
                time.sleep(0.05)
                client.send(b_txt)
                bytes_sent+=len(b_txt)
                progress_bar(bytes_sent,total_bytes)
                pass
            # 发送结束标识
            print(Fore.GREEN + "[*]Send Over!" + Fore.RESET)
            time.sleep(0.2)
            client.send("FILE_UPLOAD_OVER".encode())
            print(Fore.BLUE + client.recv(1024).decode() +
Fore.RESET)
            pass
        pass
    except:
        print("\n[*]Upload Failed!")
```

文件发送函数，开启一个套接字连接远端，然后以二进制模式读取文件，开启自定义文件名的上传，用了一个循环体和 `time.sleep` 函数，是为了防止远端接收不到客户端的数据而阻塞。

`total_bytes` 表示要发送文件的总大小, `bytes_sent` 用于表示已经发送的文件大小, 用一个循环体发送文件, 将 `total_bytes` 和 `bytes_sent` 传入进度条函数用于显示传输进度。传输完成时向服务端发送"FILE\_UPLOAD\_OVER"标识, 表示文件传输结束。

```
def client_shell_sender():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client.connect((target, port))
        print(Fore.GREEN + f"[*]connect to {target}:{port}
successfully!" + Fore.RESET)
        # 发送客户端主机的系统信息
        client.send(os.name.encode())
        while True:
            # 等待数据回转
            recv_len = 1
            resp = ""
            while recv_len:
                data = client.recv(4096).decode()
                recv_len = len(data)
                resp += data
                if recv_len < 4096:
                    break
                pass
            # 处理 windows 下的 cls
            if resp == "cls\n":
                os.system("cls")
                print(Fore.RED + "3xsh0re:>" +
Fore.RESET,end='')
            else:
                print(f"{resp}",end='')
                print(Fore.RED + "3xsh0re:>" +
Fore.RESET,end='')
            # 等待输入
            buffer = input()
            buffer += "\n"
            client.send(buffer.encode())
            pass
        except:
            client.send("user_exit\n".encode())
            print(Fore.RED + "\n[*]Exception! Exiting." +
Fore.RESET)
            client.close()
            pass
```

命令发送函数，开启一个套接字连接远端，首先发送一个当前客户端主机的系统信息，然后进入循环，等待数据回转，接收到服务端的数据，每次接收 4096 个字节的数据，然后打印，单独开了一个分支处理 clear 函数，然后等待客户端输入，输入完成后，在输入后追加\n，随后发送给服务端执行。当客户端终止程序时，会发送 user\_exit 标识。

### 3.4 流量嗅探函数设计

定义了一个 IP 类用于解析流量包，IP 类继承 Structure 类，采用了 ctypes 库定义了变量类型。

```
class IP(Structure):
    _fields_ = [
        ("ip_header_length", c_ubyte, 4),
        ("version", c_ubyte, 4),
        ("tos", c_ubyte),
        ("len", c_ushort),
        ("id", c_ushort),
        ("offset", c_ushort),
        ("ttl", c_ubyte),
        ("protocol_num", c_ubyte),
        ("sum", c_ushort),
        ("src", c_uint32),
        ("dst", c_uint32) #远程主机 IPAddr
    ]
    def __new__(self, socket_buffer=None):
        return self.from_buffer_copy(socket_buffer)
    def __init__(self, socket_buffer=None):
        # 绑定协议
        self.protocol_map = {1: "ICMP", 6: "TCP", 17: "UDP"}
        # 产生更易读的 IP 地址
        self.src_address =
socket.inet_ntoa(struct.pack("<L", self.src))
        self.dst_address =
socket.inet_ntoa(struct.pack("<L", self.dst))
        # 判断协议类型
        try:
            self.protocol =
self.protocol_map[self.protocol_num]
        pass
    except:
        self.protocol = str(self.protocol_num)
        pass
    pass
    pass
```

对于\_\_new\_\_函数, from\_buffer\_copy 是 ctypes 模块中的一个方法, 用于从一个缓冲区中复制数据到结构体对象中。它的作用是将一个二进制字符串解释成一个 C 语言中的结构体对象, 可以用来快速解析网络协议数据包的二进制数据。

对于\_\_init\_\_函数, inet\_ntoa 函数将一个 32 位的网络字节序整数转换为点分十进制的字符串格式, 然后 pack 函数按照小端字节序将整数打包成二进制数据, 再传递给 inet\_ntoa 函数进行转换。最终 self.src\_address 变量就保存了点分十进制格式的源 IP 地址。这里的<表示使用小端字节序, L 表示无符号长整数(unsigned long)的数据类型。

下面对嗅探主函数 IP\_sniffer 的进行解释:

```
def IP_sniffer():
    # 绑定本机 IP
    host = socket.gethostname()
    IPAddress = ""
    # 对于 Windows 和 Linux 的不同处理
    if (os.name == "nt"):
        IPAddress = socket.gethostbyname(host)
        print(Fore.GREEN + "[*]Your IP:" + IPAddress +
Fore.RESET)
        pass
    else:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80))
        IPAddress = s.getsockname()[0]
        s.close()
        print(Fore.GREEN + "[*]Your IP:" + IPAddress +
Fore.RESET)
    # 判断本机系统, 设置套接字
    if os.name == "nt":
        socket_protocol = socket.IPPROTO_IP
    else:
        socket_protocol = socket.IPPROTO_ICMP
    pass
    sniffer_client =
socket.socket(socket.AF_INET, socket.SOCK_RAW, socket_protocol)

    sniffer_client.bind((IPAddress, 0))

    # 设置 IP 包可以被捕捉
    sniffer_client.setsockopt(socket.IPPROTO_IP, socket.IP_HDRIN
CL, 1)

    # Windows 系统开启混杂模式, 以确保可以嗅探网卡上的所有包
```



```
if os.name == "nt":
    sniffer_client.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

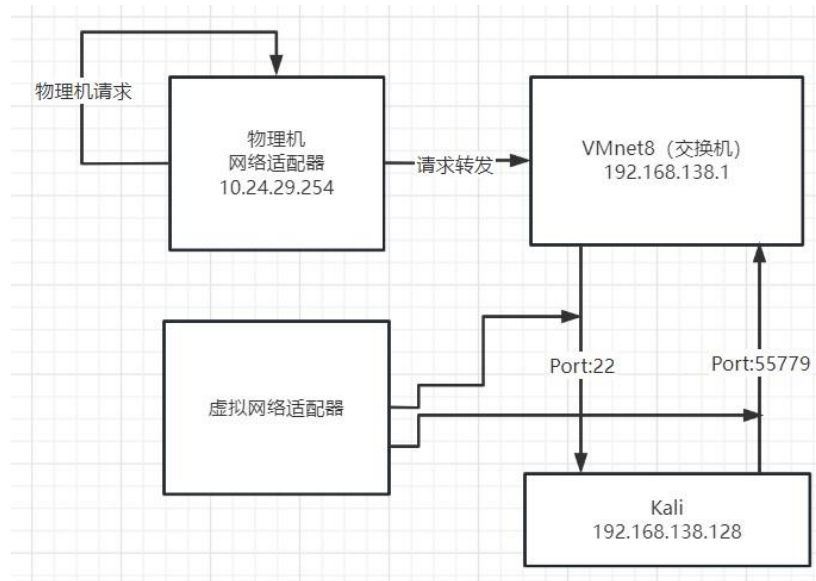
try:
    while True:
        # 读取原始包
        raw_buffer = sniffer_client.recvfrom(65565)[0]
        # 创建 IP 对象并解析
        ip_header = IP(raw_buffer[0:32])
        print(Fore.GREEN +
f"[*]Protocol:{ip_header.protocol}\tTTL:{ip_header.ttl}\t{ip_header.src_address}->{ip_header.dst_address}" + Fore.RESET)
        pass
    except KeyboardInterrupt:
        print(Fore.RED + "-"*55 + "\nyou stop sniffer!"
+Fore.RESET)
        if os.name=="nt":
            sniffer_client.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
        pass
```

首先对本机 IP 的处理，分为 Linux 和 Windows 两个分支，Windows 下直接通过 `socket.gethostbyname(host)` 获取，Linux 这样直接获取不了，采用开启一个套接字连接 Google 服务器，使用 `s.getsockname()[0]` 获取当前正在使用的网卡的 IP。由于系统原因，我们只能在 Linux 上监听到 ICMP 协议。之后 Windows 系统要开启混杂模式，以确保可以嗅探网卡上的所有包。

开启一个循环，一直读取，创建 IP 对象然后解析，打印协议类别、TTL 值、当前主机 IP 和目标地址 IP。

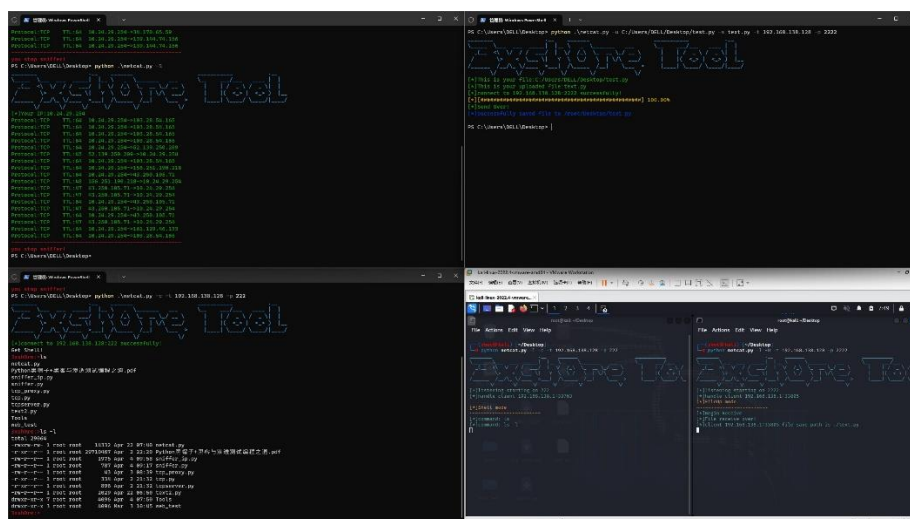
#### 4. 程序展示和测试

本地机（客户端）：Windows 11	IP: 10.24.29.254
虚拟机（服务端）：Kali 2022	IP: 192.168.138.128（NAT 模式）



1- 网络连接简示

### 三个部分的总览



2- 总览图

### 命令行模式测试:

服务端挂起: `python netcat.py -l -c -t 192.168.138.128 -p 222`

客户端连接: `python .\netcat.py -c -t 192.168.138.128 -p 222`

测试命令: `ls`, `ping www.baidu.com`, `find / -name netcat.py`, `echo 13246548 > a.txt`, `clear`

The screenshot shows a Kali Linux virtual machine with a netcat listener running on port 2222. The listener is waiting for a connection. A Windows PowerShell window is connected to the netcat listener, showing the connection details and the list of files in the current directory.

```

kali-linux-2022.4-vmware-amd64 - VMware Workstation
Workstation
File Actions Edit View Help
root@kali:~/Desktop
python netcat.py -l -U -t 192.168.138.128 -p 2222
[*] Listening starting on 2222
[*] Remote client: 192.168.138.1:13311
[*] [Shell] mode:
[*] Command: ls
[*] Command: ping www.baidu.com
[*] Command: echo 1234567 > a.txt
[*] Command: cd .;ls
[*] Command: find / -name netcat.py
[*] Command: clear

3xsh@re:~$ [connect to 192.168.138.128:2222 successfully]
Get-Shell!
3xsh@re:~$ ls
netcat.py
Python教程了+黑客与渗透测试编程之道.pdf
sniffer_ip.py
sniffer.py
tcp_proxy.py
tcp.py
tcpserver.py
Tools
web_test
3xsh@re:~$ ping www.baidu.com
PING www.a.shifen.com (110.242.68.3) 60(80) bytes of data:
64 bytes from 110.242.68.3: icmp_seq=1 ttl=128 time=56.5 ms
64 bytes from 110.242.68.3: icmp_seq=2 ttl=128 time=144 ms
64 bytes from 110.242.68.3: icmp_seq=3 ttl=128 time=59.2 ms
64 bytes from 110.242.68.3: icmp_seq=4 ttl=128 time=50.8 ms
--- www.a.shifen.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3009ms
rtt min/avg/max/ndev = 50.888/77.568/143.515/38.229 ms
3xsh@re:~$ echo 1234567 > a.txt
3xsh@re:~$ cd .;ls
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
3xsh@re:~$ find / -name netcat.py
/netcat.py
/root/Desktop/netcat.py
/root/.cache/vmware/drag_and_drop/157056/netcat.py
3xsh@re:~$

```

## 3- shell 执行 1

The screenshot shows a Kali Linux virtual machine with a netcat listener running on port 2222. The listener is waiting for a connection. A Windows PowerShell window is connected to the netcat listener, showing the connection details and the list of files in the current directory.

```

kali-linux-2022.4-vmware-amd64 - VMware Workstation
Workstation
File Actions Edit View Help
root@kali:~/Desktop
python netcat.py -l -U -t 192.168.138.128 -p 2222
[*] Listening starting on 2222
[*] Remote client: 192.168.138.1:13311
[*] [Shell] mode:
[*] Command: ls
[*] Command: ping www.baidu.com
[*] Command: echo 1234567 > a.txt
[*] Command: cd .;ls
[*] Command: find / -name netcat.py
[*] Command: clear

3xsh@re:~$ [connect to 192.168.138.128:2222 successfully]
Get-Shell!
3xsh@re:~$ ls
netcat.py
Python教程了+黑客与渗透测试编程之道.pdf
sniffer_ip.py
sniffer.py
tcp_proxy.py
tcp.py
tcpserver.py
Tools
web_test
3xsh@re:~$ ping www.baidu.com
PING www.a.shifen.com (110.242.68.3) 60(80) bytes of data:
64 bytes from 110.242.68.3: icmp_seq=1 ttl=128 time=56.5 ms
64 bytes from 110.242.68.3: icmp_seq=2 ttl=128 time=144 ms
64 bytes from 110.242.68.3: icmp_seq=3 ttl=128 time=59.2 ms
64 bytes from 110.242.68.3: icmp_seq=4 ttl=128 time=50.8 ms
--- www.a.shifen.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3009ms
rtt min/avg/max/ndev = 50.888/77.568/143.515/38.229 ms
3xsh@re:~$ echo 1234567 > a.txt
3xsh@re:~$ cd .;ls
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
3xsh@re:~$ find / -name netcat.py
/netcat.py
/root/Desktop/netcat.py
/root/.cache/vmware/drag_and_drop/157056/netcat.py
3xsh@re:~$

```

## 4- shell 执行 2

文件上传模式测试:

服务端挂起: `python netcat.py -l -U -t 192.168.138.128 -p 2222`

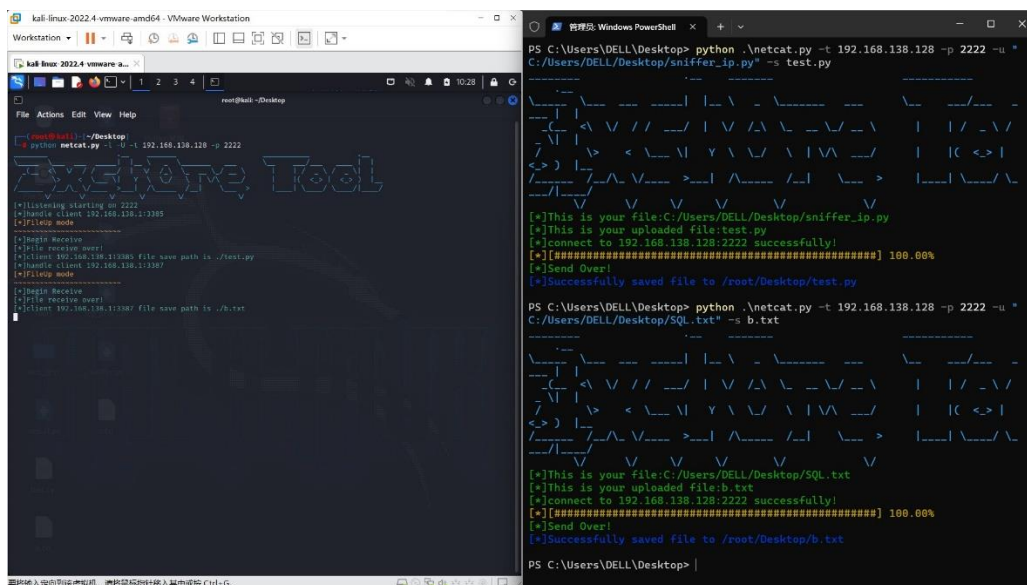
客户端连接:

`python .\netcat.py -t 192.168.138.128 -p 2222 -u`

`"C:/Users/DELL/Desktop/sniffer_ip.py" -s test.py`

`python .\netcat.py -t 192.168.138.128 -p 2222 -u`

`"C:/Users/DELL/Desktop/SQL.txt" -s b.txt`



The screenshot shows a Kali Linux virtual machine with a netcat listener running on port 2222. The listener is in 'File mode' and is ready to receive files. A Windows PowerShell terminal shows a netcat client connecting to 192.168.138.128 on port 2222. The client uploads a file named 'test.py' and then 'b.txt'. The listener successfully receives both files and saves them to the root directory.

```
PS C:\Users\DELL\Desktop> python .\netcat.py -t 192.168.138.128 -p 2222 -u *
C:/Users/DELL/Desktop/sniffer_ip.py" -s test.py

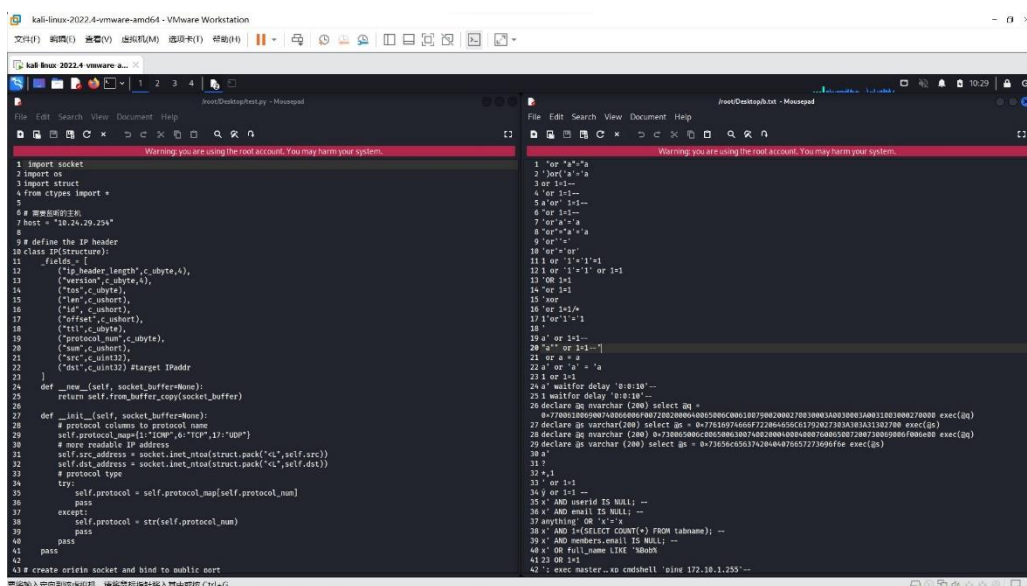
-----
[+]This is your file:C:/Users/DELL/Desktop/sniffer_ip.py
[+]This is your uploaded file:test.py
[+]connect to 192.168.138.128:2222 successfully!
[+]##### 100.00%
[+]Send Over!
[+]Successfully saved file to /root/Desktop/test.py

PS C:\Users\DELL\Desktop> python .\netcat.py -t 192.168.138.128 -p 2222 -u *
C:/Users/DELL/Desktop/SQL.txt" -s b.txt

-----
[+]This is your file:C:/Users/DELL/Desktop/SQL.txt
[+]This is your uploaded file:b.txt
[+]connect to 192.168.138.128:2222 successfully!
[+]##### 100.00%
[+]Send Over!
[+]Successfully saved file to /root/Desktop/b.txt

PS C:\Users\DELL\Desktop>
```

5- 文件上传 1



The screenshot shows a Kali Linux virtual machine with a netcat listener running on port 2222. The listener is in 'File mode' and is ready to receive files. A Windows PowerShell terminal shows a netcat client connecting to 192.168.138.128 on port 2222. The client uploads a file named 'test.py' and then 'b.txt'. The listener successfully receives both files and saves them to the root directory.

```
1 import socket
2 import os
3 import struct
4 from ctypes import *
5
6 # 需要知道的主机
7 host = "192.168.138.128"
8
9 # 定义 IP 的 header
10 class IP(struct):
11     _fields_ = [
12         ("ip_header_length", c_ubyte, 4),
13         ("version", c_ubyte, 4),
14         ("tos", c_ubyte, 4),
15         ("len", c_ushort, 4),
16         ("id", c_ushort, 4),
17         ("offset", c_ushort, 4),
18         ("ttl", c_ubyte, 4),
19         ("protocol_num", c_ubyte, 4),
20         ("sum", c_ushort, 4),
21         ("src", c_uint32, 4),
22         ("dst", c_uint32, 4)
23     ]
24
25 def _new(self, socket_buffer=None):
26     return self.from_buffer_copy(socket_buffer)
27
28 def _init(self, socket_buffer=None):
29     # protocol columns to protocol name
30     self.protocol_map["TCP"] = "TCP"
31     # more readable IP address
32     self.src_address = socket.inet_ntoa(struct.pack("<L", self.src))
33     self.dst_address = socket.inet_ntoa(struct.pack("<L", self.dst))
34     # protocol type
35     try:
36         self.protocol = self.protocol_map[self.protocol_num]
37     except:
38         self.protocol = str(self.protocol_num)
39     pass
40
41 # create origin socket and bind to public port
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

6- 文件上传 2

嗅探模式测试:

Windows 和 Linux: python netcat.py -S

虚拟上可以另开一个终端, ping www.sina.com, 否则很少会主动出现流量

