

湖南人文科技学院

网络安全课程设计

|  |  |
| --- | --- |
| 项目名称： | **网络嗅探器** |
| 学院名称： | **信息学院** |
| 组 长： | **19420203曾思龙** |
| 组 员： | **19420215李锦程**  **19420229田威** |
| 专业年级： | **网络工程2019级** |
| 指导教师： | **彭剑** |

湖南人文科技学院信息学院

目录

[1. 项目内容 3](#_Toc605)

[2. 引言 3](#_Toc31180)

[3. 相关技术 4](#_Toc19999)

[3.1 嗅探原理 4](#_Toc3831)

[4. 需求分析 5](#_Toc2539)

[4.1 任务概述 5](#_Toc15303)

[4.2 系统功能需求 5](#_Toc27768)

[4.3 功能模块图 6](#_Toc15429)

[4.4 数据包过滤规则 6](#_Toc18736)

[5.主要功能的代码实现 7](#_Toc1963)

[5.1 实时抓包功能实现 7](#_Toc19954)

[5.2 pcap文件解析 9](#_Toc8616)

[5.3 过滤数据包 13](#_Toc28630)

[6.界面设计 15](#_Toc12938)

[6.1 首页显示 15](#_Toc1827)

[6.2 数据包显示 16](#_Toc1513)

[7.操作手册 16](#_Toc23508)

[7.1 首页 16](#_Toc19343)

[7.2 选择打开文件之后 17](#_Toc12123)

[7.3 选择网卡抓包之后 17](#_Toc5291)

[7.4 停止抓包 18](#_Toc30097)

[7.5 保存文件 18](#_Toc6033)

1. **项目内容**

完成一个网络嗅探器的桌面程序，具备嗅探器功能和分析器功能。

1. **引言**

随着网络技术的飞速发展，网络安全问题越来越被人重视。嗅探技术作为网络安全攻防中最基础的技术，既可以用于获取网络中传输的大量敏感信息，也可以用于网络管理。通过获取网络数据包的流向和内容等信息，可以进行网络安全分析和网络威胁应对。

本课程设计通过分析网络上常用的嗅探器软件，在了解其功能和原理的基础上，以PyCharm为开发平台，使用第三方库scapy，按照软件工程的思想进行设计并实现了一个网络嗅探工具。该嗅探工具的总体架构划分为5部分，分别是最底层的数据缓存和数据访问，中间层的数据捕获，协议过滤，协议分析和最顶层的图形画用户界面。

本嗅探器工具完成了：1.数据包捕获及分析2.协议过滤的主要功能。实现了对网络协议，源IP地址，目标IP地址及端口号等信息的显示，使得程序能够比较全面地分析出相关信息以供用户参考决策。

1. **相关技术**

**3.1 嗅探原理**

嗅探器是一种常用的收集有用数据的方法,嗅探器是利用计算机的网络接口也就是网卡，截获目的地为其它计算机的数据报文的一种工具。

由于组网方式不同，嗅探器的工作原理也有所不同，总共可以分为三类:

(1)基于网卡混杂模式的嗅探原理;

(2)基于ARP欺骗的网络嗅探原理;

(3)基于中间人攻击的嗅探原理。

本次课程设计就是利用第一种方式来实现嗅探器的功能，可见监听到所有流经同一以太网网段的数据包，这是一种被动嗅探的方式。

1. **需求分析**

**4.1 任务概述**

小程序需要实习分析器和嗅探器俩个功能，以及可视化界面的设计。用户可以打开pcap文件进行分析数据，也可以实时抓包，获取流量数据。

**4.2 系统功能需求**

**4.2.1 可视化操作功能**

用户打开程序可查看当前电脑的所有网卡。点击网卡进行实时抓包。同时也可以选择打开文件进行文件的流量数据分析。在实时抓包和文件流量分析的界面查看每条数据的抓包时间，源端口，源地址，目的地址，目的端口，协议类型，每条数据可查看每层协议的信息以及二进制数据。

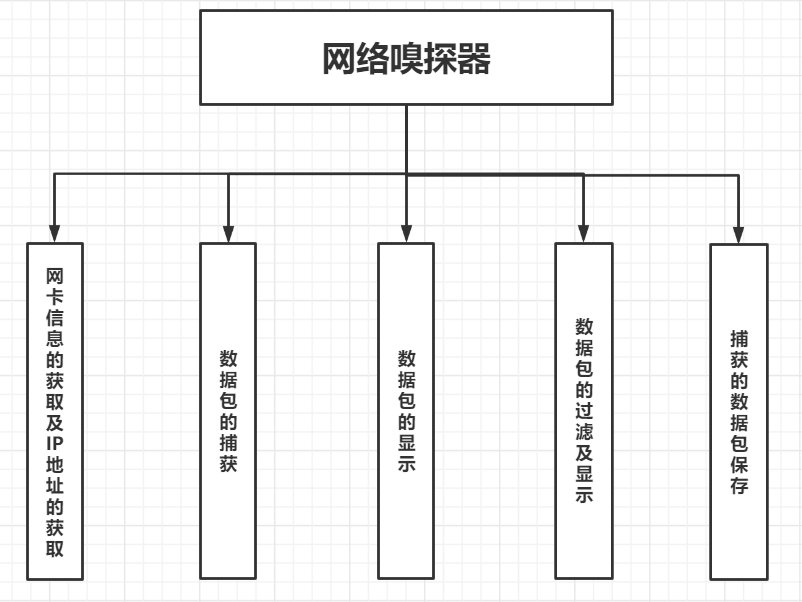
**4.2.2 嗅探器功能**

用户选择网卡后，进行数据抓包，并可视化显示。在抓包的过程中，用户可选择停止抓包，停止抓包后进行重新抓包，停止抓包后进行保存成pcap文件。

**4.2.3 分析器功能**

将得到的数据逐行显示，并将数据与其含义对应显示，并将每一层的数据逐层显示。用户可输入特定字符串进行过滤数据包，查看自己想要的数据包。

**4.3 功能模块图**



**4.4 数据包过滤规则**

|  |  |
| --- | --- |
| 字符规则 | 描述 |
| ip==xx.xx.xx.xx | 匹配所有的源端或者目的端ip是此ip的数据包进行过滤显示 |
| port==xxx | 匹配所有的源端或者目的端端口值是此端口值的数据包进行过滤显示 |
| Src.ip ==xx.xx.xx.xx | 匹配源端ip是此ip的数据包进行过滤显示 |
| Dis.ip==xx.xx.xx.xx | 匹配目的端ip是此ip的数据包进行过滤显示 |
| src.port==12 | 匹配源端端口是此端口的数据包进行过滤显示 |
| dst.port==12 | 匹配目的端端口是此端口的数据包进行过滤显示 |
| tcp.port==12 | 匹配tcp协议数据包并且端口是此端口的数据包进行过滤显示 |
| Udp.port==12 | 匹配udp协议数据包并且端口是此端口的数据包进行过滤显示 |
| tcp.stream==12 | 匹配tcp协议数据包并且数据流是此值的数据包进行过滤显示 |
| udp.stream==12 | 匹配udp协议数据包并且数据流是此值的数据包进行过滤显示 |

**5.主要功能的代码实现**

**5.1 实时抓包功能实现**

class mySniffer:

def \_\_init\_\_(self):

self.iface = None

self.socket = None

@staticmethod

def show\_all\_ifaces(print\_res=True):

"""打印所有interface"""

return scapy.IFACES.show(print\_result=print\_res)

def create\_socket(self, index):

"""根据打印的索引确定一个interface，然后创建socket绑定用于抓包"""

if index < 0:

return False

self.iface = scapy.IFACES.dev\_from\_index(index)

# 进行绑定，便于抓包

# scapy.conf.iface.setmonitor(True)

self.socket = scapy.conf.L2socket(iface=self.iface)

return True

def get\_one\_packet(self):

""" 抓取一个数据包

调用这个函数前需要使用create\_socket创建socket

返回依次为:链路层数据包类型、数据包数据、时间

:return: 链路层数据包类型，数据包，时间

"""

# 调用这个函数 抓取一个数据包

return self.socket.recv\_raw()

5.2 pcap文件解析

def parse\_a\_packet(packet, info, packet\_head\_json, dns\_stream, dns\_stream\_index):

""" 解析一个数据包，最后返回info和json

"""

# 解析数据包的链路层

ip\_packet, eth\_header = parse\_eth(packet)

info['src\_addr'] = eth\_header['Source']

info['dst\_addr'] = eth\_header['Destination']

info['type'] = 'Ethernet'

packet\_head\_json['Ethernet'] = eth\_header

if eth\_header['Type'] == '0x0800':

trans\_packet, ip\_header = parse\_ipv4(ip\_packet)

info['src\_addr'] = ip\_header['Source\_Address']

info['dst\_addr'] = ip\_header['Destination\_Address']

info['type'] = 'IPv4'

packet\_head\_json['Internet Protocol Version 4'] = ip\_header

if ip\_header['Protocol'] == '6':

# 解析tcp

app\_packet, tcp\_header = parse\_tcp(trans\_packet)

info['src\_port'] = tcp\_header['Source\_Port']

info['dst\_port'] = tcp\_header['Destination\_Port']

info['type'] = 'TCP'

packet\_head\_json['Transmission Control Protocol'] = tcp\_header

elif ip\_header['Protocol'] == '17':

# 解析udp

app\_packet, udp\_header = parse\_udp(trans\_packet)

info['src\_port'] = udp\_header['Source\_Port']

info['dst\_port'] = udp\_header['Destination\_Port']

info['type'] = 'UDP'

packet\_head\_json['User Datagram Protocol'] = udp\_header

if info['dst\_port'] == '53':

# 发送DNS请求

# 格式：流序号-本机端口号-dns服务器ip

dns\_stream.append(str(dns\_stream\_index) + '-' + info['src\_port'] + '-' + info['dst\_addr'])

info['dns\_stream'] = dns\_stream\_index

info['type'] = 'DNS'

dns\_stream\_index += 1

if info['src\_port'] == '53':

# 收到DNS应答

for item in dns\_stream:

index, port, ip = item.split('-')

if port == info['dst\_port'] and ip == info['src\_addr']:

info['dns\_stream'] = index

info['type'] = 'DNS'

dns\_stream.remove(item)

break

elif ip\_header['Protocol'] == '1':

# 解析icmp

icmp\_header = parse\_icmp(trans\_packet)

info['type'] = 'ICMP'

packet\_head\_json['ICMP'] = icmp\_header

# else:

# 其他类型的协议，未实现

# print("无法解析IP层头部的字段Protocol(" + ip\_header['Protocol'] + ')')

elif eth\_header['Type'] == '0x0806':

arp\_header = parse\_arp(ip\_packet)

info['type'] = 'ARP'

packet\_head\_json['ARP'] = arp\_header

elif eth\_header['Type'] == '0x86dd':

pkt, ip\_header = parse\_ipv6(ip\_packet)

info['type'] = 'IPv6'

info['src\_addr'] = ip\_header['Source\_Address']

info['dst\_addr'] = ip\_header['Destination\_Address']

packet\_head\_json['Internet Protocol Version 6'] = ip\_header

elif eth\_header['Type'] == '0x8864':

print("链路层无法识别[PPPoE]协议")

elif eth\_header['Type'] == '0x8100':

print("链路层无法识别[802.1Q tag]协议")

elif eth\_header['Type'] == '0x8847':

print("链路层无法识别[MPLS Label]协议")

else:

# unknown ip protocol

print("链路层无法识别")

return info, packet\_head\_json, dns\_stream, dns\_stream\_index

**5.3 过滤数据包**

def parse\_filter(self, filter\_str):

"""解析输入过滤器的字符串是否符合语法，符合则返回true"""

if filter\_str == '':

return 0

# 去除所有空格

filter\_str = filter\_str.replace(' ', '')

if filter\_str == 'tcp':

return 1

elif filter\_str == 'udp':

return 2

elif filter\_str == 'dns':

return 13

else:

filter\_str = filter\_str.split('==')

# ip==1.1.1.1

if filter\_str[0] == 'ip':

return 3

# port==23

elif filter\_str[0] == 'port':

return 4

# src.ip==1.1.1.1

elif filter\_str[0] == 'src.ip':

return 5

# dst.ip==1.1.1.1

elif filter\_str[0] == 'dst.ip':

return 6

# src.port==12

elif filter\_str[0] == 'src.port':

return 7

# dst.port==12

elif filter\_str[0] == 'dst.port':

return 8

# tcp.port==12

elif filter\_str[0] == 'tcp.port':

return 9

# udp.port==12

elif filter\_str[0] == 'udp.port':

return 10

# tcp.stream==12

elif filter\_str[0] == 'tcp.stream':

return 11

# udp.stream==12

elif filter\_str[0] == 'udp.stream':

return 12

return -1

**6.界面设计**

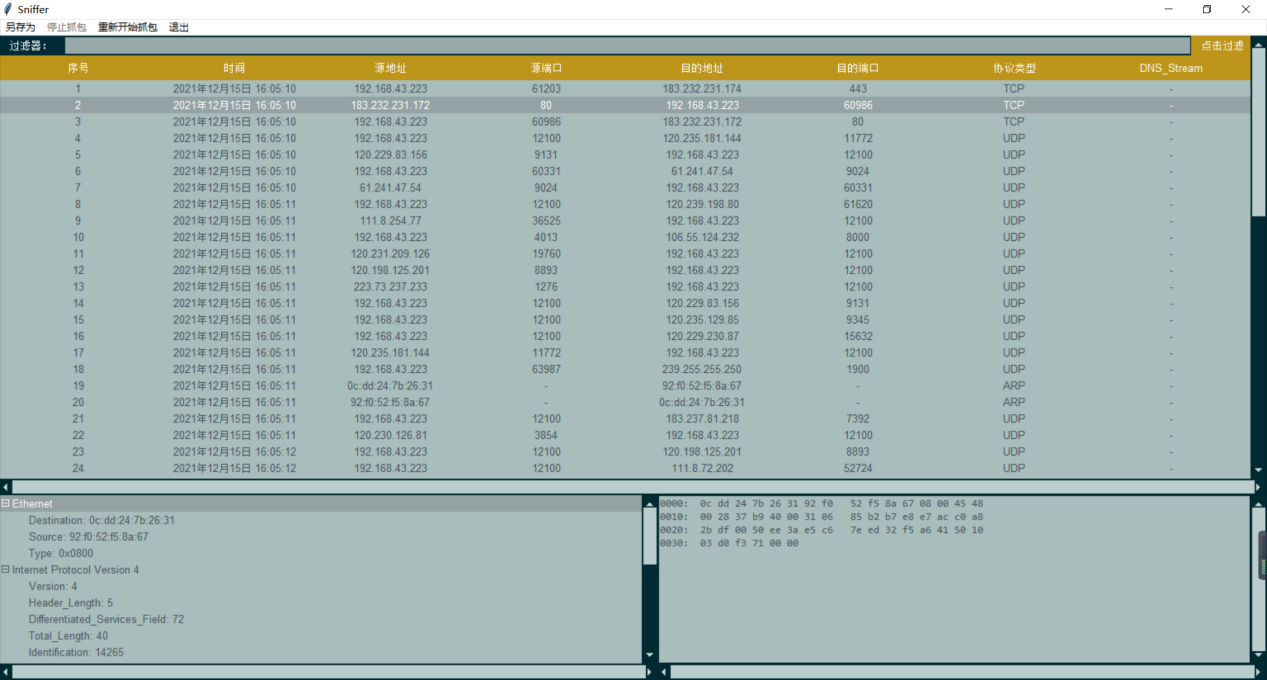
**6.1 首页显示**

可点击网卡进行实时抓包，或者打开



**6.2 数据包显示**

数据显示，以及分析



**7.操作手册**

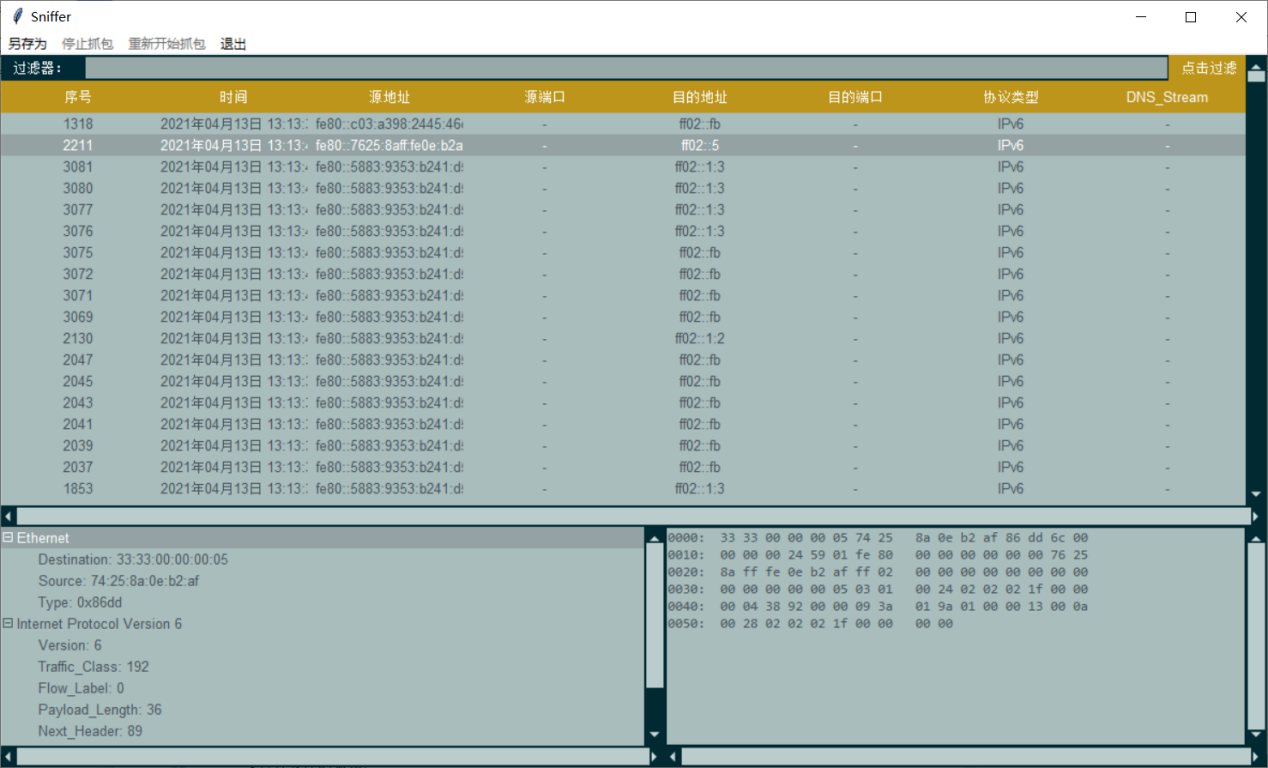
**7.1 首页**

用户可选择打开pcap文件或者点击网卡进行实时抓包



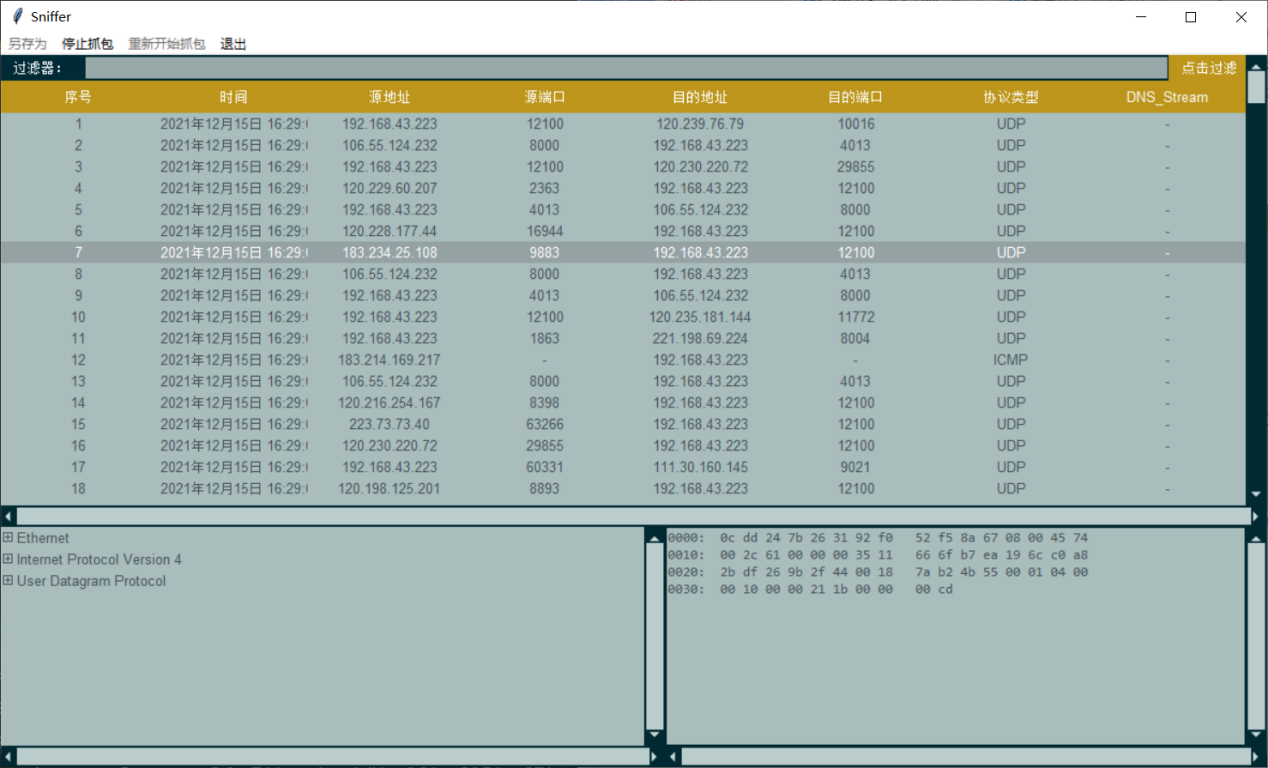
**7.2 选择打开文件之后**

可查看文件的数据



**7.3 选择网卡抓包之后**

数据包实时显示



**7.4 停止抓包**

点击停止抓包后可选择重新抓包或者另存为一个pcap文件



**7.5 保存文件**

