# 实验1A：带GUI的网络嗅探器设计与实现

Lihaodong22@mails.ucas.ac.cn

## 实验目的

设计并且实现一个带GUI的网络嗅探器，实现对网卡的抓包和解包功能。

## 实验要求

1.实验中必须实现 NPS 功能

2.实验中尽量完善 NPA 功能

## 实验环境

1. 操作系统：win10
2. 开发环境：PyCharm Community Edition 2022.3.3
3. Python版本：Python 3.6

## 准备阶段

1. 导包

import Hexadecimal  
import Resolution  
import pcap  
import qdarkstyle  
from PyQt5.QtWidgets import \*  
from PyQt5.QtGui import QIcon, QFont  
from scapy.all import \*

Hexadecimal是显示十六进制信息界面的包；Resolution是解析数据界面的包；pcap是一个数据包抓取库；qdarkstyle是一个适用于QT应用程序的最完整的深色样式库；PyQT5是一个用于设计交互界面GUI的一个库；scapy是用来抓取数据和解析底层网络数据包的一个库。

1. 根据使用者的设备，列出当前操作系统的所有网络接口

devs = pcap.findalldevs()

1. 将所有网络接口都设置成混杂模式，设置数据包过滤器

for idx,item in enumerate(devs):  
 pc = pcap.pcap(devs[idx], promisc=True, immediate=True, timeout\_ms=50)  
 pc.setfilter('tcp port 80')

## 实验步骤

### 5.1主界面的设计与实现

主界面由PYQT5的组件搭建而成。在主界面，我主要设计与实现了五个功能：1.开始抓包；2.清空数据；3.十六进制信息；4.解析数据；5.退出系统。与1、3、4功能相关的有一个下拉列表选择框和两个输入框，分别是：网卡名称、抓包数量、查看/解析序号。如下图所示。



##### 5.1.1开始抓包的设计与实现

**操作流程：**首先，需要先选择“网卡名称”和输入“抓包数量”，然后点击“开始抓包”，便可将抓到的包显示在界面上，否则会有相应的提示信息弹出。

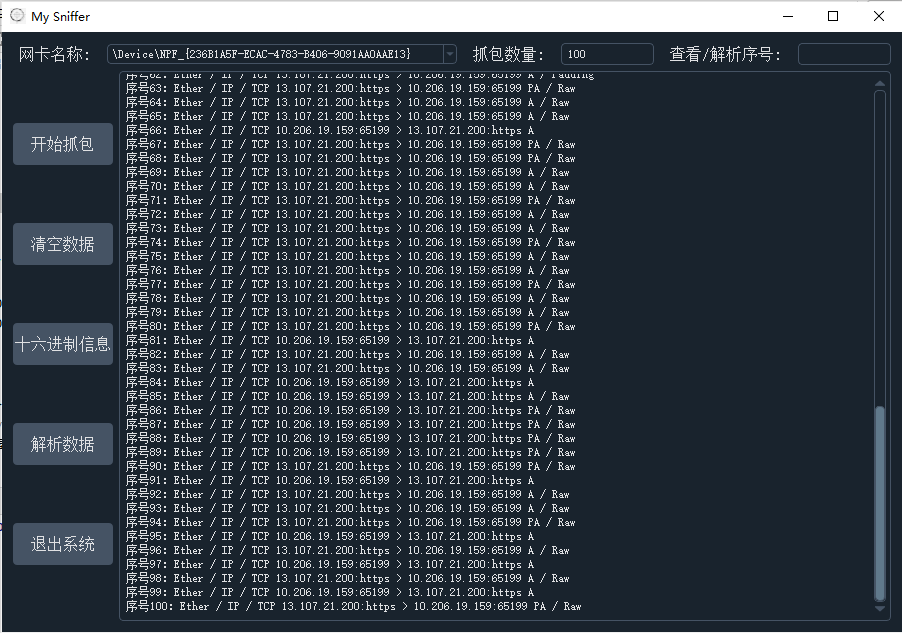
**设计思路：**在点击“开始抓包”的时候，先要检查是否选择网卡名称和填入抓包数量，只有在两者都正确的填入的情况下，才会启动开始抓包。我们使用scapy包里的sniffer进行抓包，将网卡名称net\_card\_input和输入抓包的数量quantity\_input传入sniffer即可抓包。

packets=sniff(iface=net\_card\_input.currentText(),count=int(quantity\_input.text())) .抓到包后，我们使用一个for循环将每条数据依次显示在抓包界面上。

**主要代码实现：**

def start\_captureClicked(self):  
 if net\_card\_input.currentText() == '--请选择--':  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请选择网卡名称！')  
 msg\_box.exec\_()  
 elif quantity\_input.text() == '': # 没有输入抓包数量  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请输入抓包数量！')  
 msg\_box.exec\_()  
 else:  
 idx = int(quantity\_input.text()) # 抓包数量  
 if idx > 0:  
 global packets  
 self.data\_area.clear() # 清空抓包数据  
 packets = sniff(iface=net\_card\_input.currentText(), count=int(quantity\_input.text())) # 抓包  
 # print(packets)  
 # 显示抓包数据f  
 count = 1  
 for p in packets:  
 # print(p)  
 self.data\_area.append('序号'+str(count)+'：'+str(p))  
 count += 1  
 else:  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请输入一个正整数！')  
 msg\_box.exec\_()

**运行结果截图：**



##### 5.1.2清空数据的设计与实现

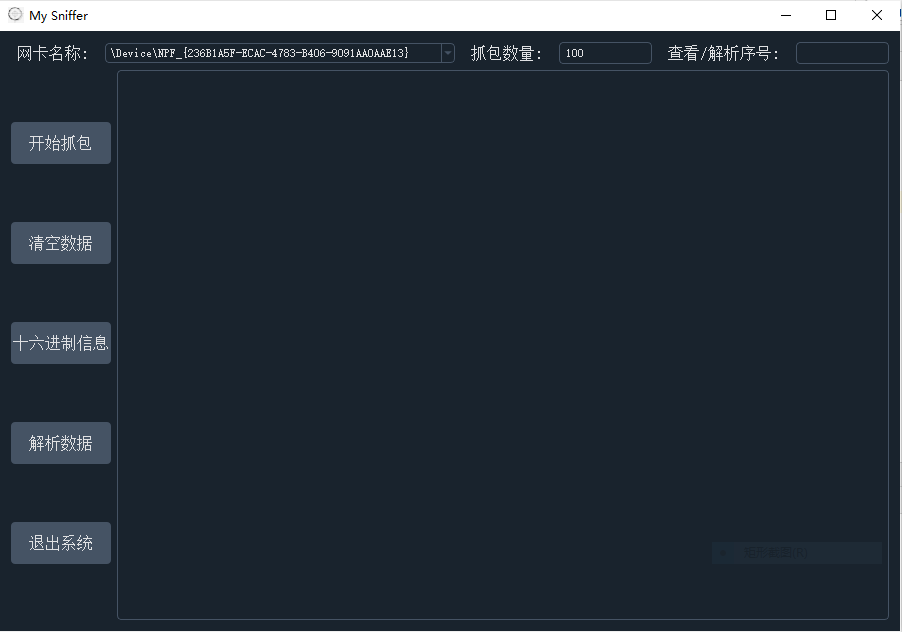
**操作流程：**点击“清空数据”即可将界面之前抓到的包全部清空。

**设计思路：**点击“清空数据”后，抓包区域data\_area调用clear()函数即可实现清空。

**主要代码实现：**

def clear\_dataClicked(self):  
 self.data\_area.clear() # 清空抓包数据

**运行结果截图：**

****

##### 5.1.3十六进制信息的设计与实现

**操作流程：**在已经抓到包的前提下，当我们输入“查看/解析序号”时，然后再点击“十六进制信息”，便可弹出一个界面，显示我们所查询序号数据的十六进制。

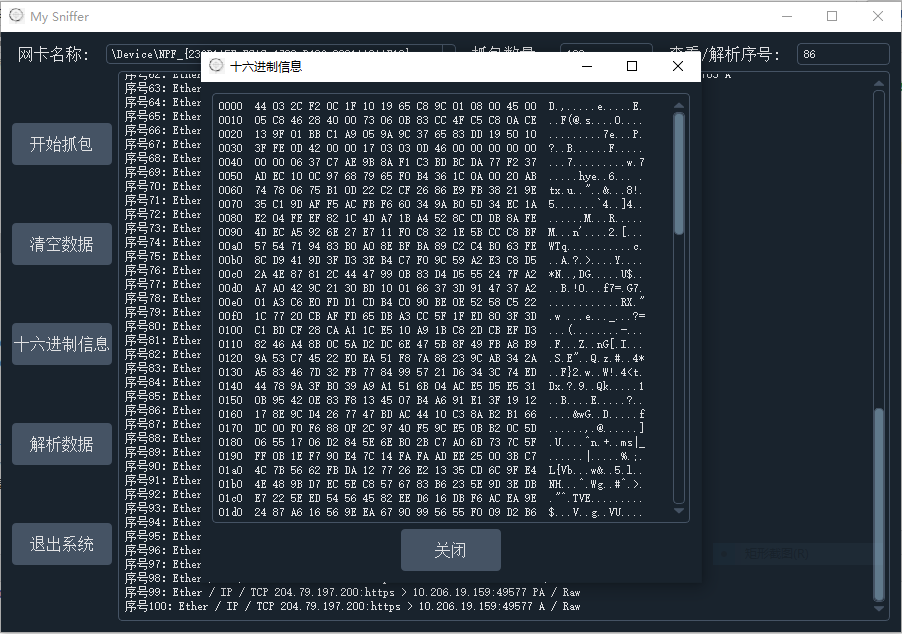
**设计思路：**当输入“查看/解析序号”时，如果该序号在合理的范围内，我们将使用该序号查找数据，我们在for循环里依次遍历，直到找到该数据执行break跳出循环。我们通过hexdump()函数将其变为十六进制，然后写入out\_hex.txt文件。之后创建一个查看窗口对象self.hexadecimal = Hexadecimal.Hexadecimal()，从out\_hex.txt文件中读出该十六进制数据显示出来，再销毁out\_hex.txt文件即可。

**主要代码实现：**

def binary\_informationClicked(self):  
 if index\_input.text() == '': # 没有输入查看序号  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请输入查看序号！')  
 msg\_box.exec\_()  
 else:  
 # if quantity\_input.text() == '':  
 if self.data\_area.toPlainText() == '': # 如果抓包显示区域已被清空，即没有包  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请先抓包！')  
 msg\_box.exec\_()  
 else:  
 idx = int(index\_input.text()) # 查看序号  
 if idx > 0 and idx <= int(quantity\_input.text()):  
 # print(idx)  
 # savedDataOut = sys.stdout # 保存标准输出流  
 # 创建文件out\_hex.txt  
 with open('out\_hex.txt', 'w+', encoding='UTF-8') as file:  
 sys.stdout = file # 标准输出重定向至文件  
 for k, p in enumerate(packets): # 遍历抓包数据  
 if k + 1 == idx: # 找到要查看的那条数据  
 print(hexdump(p))

break  
 # sys.stdout = savedDataOut # 恢复标准输出流  
  
 self.hexadecimal = Hexadecimal.Hexadecimal() # 创建查看窗口对象  
 self.hexadecimal.show()  
 # sys.exit(app.exec\_())  
 else:  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请输入正确的查看序号！')  
 msg\_box.exec\_()

**运行结果截图：**

****

##### 5.1.4解析数据的设计与实现

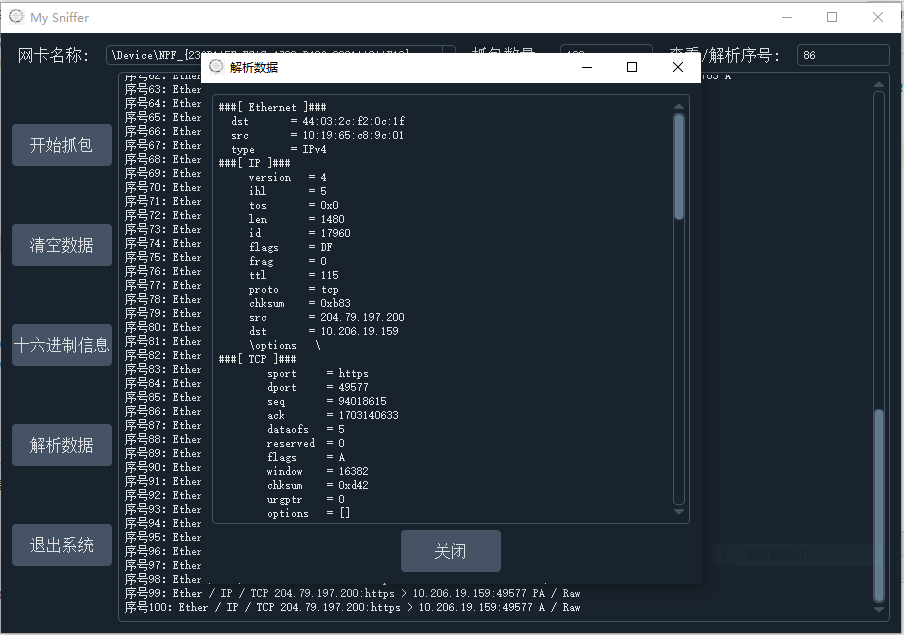
**操作流程：**在已经抓到包的前提下，当我们输入“查看/解析序号”时，然后再点击“解析数据”，便可弹出一个界面，显示我们所解析序号的解析数据。

**设计思路：**当输入“查看/解析序号”时，如果该序号在合理的范围内，我们将使用该序号查找数据，我们在for循环里依次遍历，直到找到该数据执行break跳出循环。然后写入out\_res.txt文件。之后创建一个解析窗口对象self.resolution = Resolution.Resolution()，从out\_res.txt文件中读出该解析数据显示出来，再销毁out\_res.txt文件即可。

**主要代码实现：**

def parse\_dataClicked(self):  
 if index\_input.text() == '': # 没有输入解析序号  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请输入解析序号！')  
 msg\_box.exec\_()  
 else:  
 # if quantity\_input.text() == '':  
 if self.data\_area.toPlainText() == '': # 如果抓包显示区域已被清空，即没有包  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请先抓包！')  
 msg\_box.exec\_()  
 else:  
 idx = int(index\_input.text()) # 解析序号  
 if idx > 0 and idx <= int(quantity\_input.text()):  
 # print(idx)  
 # savedDataOut = sys.stdout # 保存标准输出流  
 # 创建文件out.txt  
 with open('out\_res.txt', 'w+', encoding='UTF-8') as file:  
 sys.stdout = file # 标准输出重定向至文件  
 for k, p in enumerate(packets): # 遍历抓包数据  
 if k + 1 == idx: # 找到要解析的那条数据  
 print(p.show())  
 break  
 self.resolution = Resolution.Resolution() # 创建解析窗口对象  
 self.resolution.show()  
 else:  
 msg\_box = QMessageBox(QMessageBox.Warning, '警告', '请输入正确的解析序号！')  
 msg\_box.exec\_()

**运行结果截图：**



#### 5.1.5退出系统的设计与实现

**操作流程：**点击“退出系统”即可退出该嗅探器系统。

**设计思路：**点击“退出系统”后，该应用对象QApplication.instance()调用quit()函数即可退出该系统。

**主要代码实现：**

def exit\_sysClicked(self):  
 app = QApplication.instance()  
 # 退出应用程序  
 app.quit()

**运行结果截图：**一键退出系统，无截图。

### 5.2解析界面的设计与实现

解析界面由PYQT5的组件搭建而成。在功能方面，主要实现了两个功能：

* 显示解析数据：从out\_res.txt文件读出解析数据显示出来，销毁out\_res.txt文件；
* 关闭该界面：点击“关闭”按钮后，调用hide()函数关闭该界面。

界面截图：



### 5.3十六进制信息界面的设计与实现

十六进制信息界面由PYQT5的组件搭建而成。在功能方面，主要实现了两个功能：

* 显示十六进制信息：从out\_hex.txt文件读出解析数据显示出来，销毁out\_hex.txt文件；
* 关闭该界面：点击“关闭”按钮后，调用hide()函数关闭该界面。

界面截图：



### 实验讨论和分析

在本次实验的过程中，我主要分为三个步骤来完成：第一步，构思我要完成的嗅探器，具体包括界面怎么设计、实现哪些功能等；第二步，代码实现我的嗅探器，这是最耗时的一步，在这个过程中，我自学了PYQT5、pcap和scapy；第三步，对所实现的嗅探器进行优化和微调，修改了某些功能的实现方式，之前的实现思路和代码也都以注释的形式保存在源码中。

在该网络嗅探器的设计过程中，我充分考虑了该系统的健壮性，大量完善了各种提示信息和错误信息的处理，通过这些提示信息的弹出框也可以指引用户去正确地操作该平台。

### 七、总结与展望

本次实验我设计和实现了带GUI的网络嗅探器，主要实现了五个功能：开始抓包、清空数据、十六进制信息、解析数据和退出系统，基本完成了本次实验的基本要求：抓包和解包功能。但因为时间有限，该网络嗅探器还有一些需要继续加工和完善的地方，譬如：抓包和解包后格式的显示，可以通过表格的形式来显示，那样就可以实现的批次的筛选，同时通过表格的方式呈现也可以实现更多的功能；此外，我设计网络嗅探器外观也需要继续的改善，虽然该嗅探器已经实现了基本功能，但与商用的抓包工具还有有不小差距，我会在以后的时间里继续完善，争取做一个自己的产品出来。