**流量分析**

1. **流量分析简介**

CTF 比赛中, 流量包的取证分析是另一项重要的考察方向。

通常比赛中会提供一个包含流量数据的 PCAP 文件，有时候也会需要选手们先进行修复或重构传输文件后，再进行分析。

PCAP 这一块作为重点考察方向，复杂的地方在于数据包里充满着大量无关的流量信息，因此如何分类和过滤数据是参赛者需要完成的工作。

总的来说有以下几个步骤

* 总体把握
  + 协议分级
  + 端点统计
* 过滤筛选
  + 过滤语法
  + Host，Protocol，contains，特征值
* 发现异常
  + 特殊字符串
  + 协议某字段
  + flag 位于服务器中
* 数据提取
  + 字符串取
  + 文件提取

总的来说比赛中的流量分析可以概括为以下三个方向:

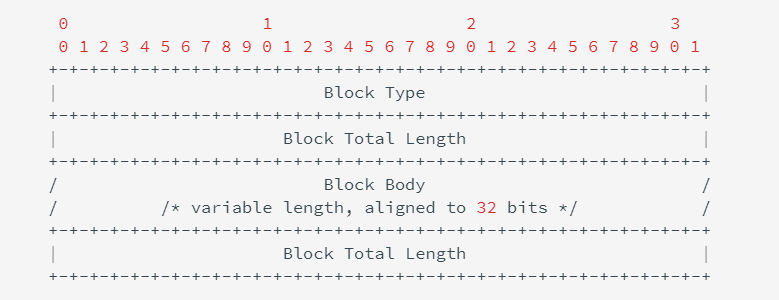
* 流量包修复
* 协议分析
* 数据提取

1. **PCAP 文件修复**

## PCAP 文件结构

一般来说, 对于 PCAP 文件格式考察较少，且通常都能借助于现成的工具如 pcapfix 直接修复，这里大致介绍下几个常见的块，详细可以翻看 [Here](http://www.tcpdump.org/pcap/pcap.html)。

* Tools
  + [PcapFix Online](https://f00l.de/hacking/pcapfix.php)
  + [PcapFix](https://github.com/Rup0rt/pcapfix/tree/devel)

一般文件结构

目前所定义的常见块类型有

1. Section Header Block: it defines the most important characteristics of the capture file.
2. Interface Description Block: it defines the most important characteristics of the interface(s) used for capturing traffic.
3. Packet Block: it contains a single captured packet, or a portion of it.
4. Simple Packet Block: it contains a single captured packet, or a portion of it, with only a minimal set of information about it.
5. Name Resolution Block: it defines the mapping from numeric addresses present in the packet dump and the canonical name counterpart.
6. Capture Statistics Block: it defines how to store some statistical data (e.g. packet dropped, etc) which can be useful to undestand the conditions in which the capture has been made.

## 常见块

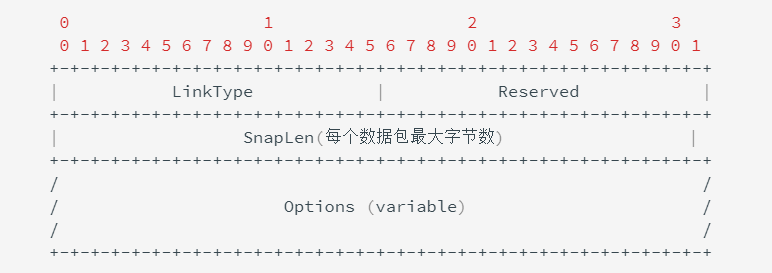
### **Section Header BlocK(文件头)**

必须存在, 意味着文件的开始



### Interface Description Block(接口描述)

必须存在, 描述接口特性



### Packet Block(数据块)



## 例题

题目：第一届 “百度杯” 信息安全攻防总决赛 线上选拔赛：find the flag

WP：<https://www.cnblogs.com/ECJTUACM-873284962/p/9884447.html>

首先我们拿到这样一道流量包的题目，题目名称为 find the flag 。这里面给了很多提示信息，要我们去找到 flag 。

**第一步，搜索**flag**字样**

我们先去搜索看看流量包里面有没有 flag 。我们使用 strings 命令去找一下流量包， Windows的朋友可以用 notepad++ 的搜索功能去寻找。

搜索命令如下：

strings findtheflag.cap | grep flag

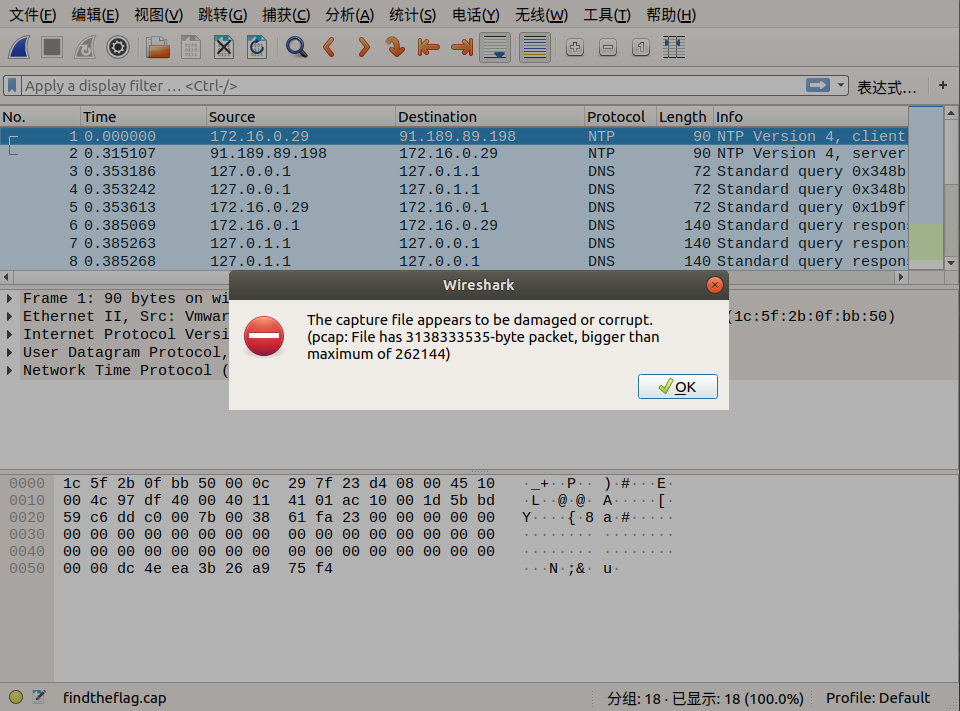
搜索结果如下：



我们发现搜出了一大堆的东西，我们通过管道去过滤出 flag 信息，似乎没有发现我们所需要找的答案。

**第二步，流量包修复**

我们用 wireshark 打开这个流量包

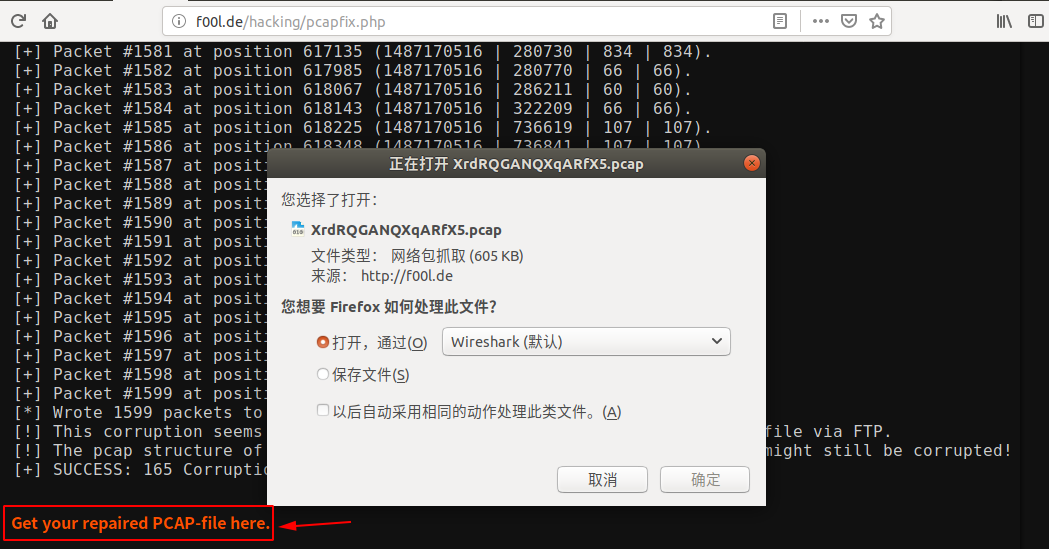


我们发现这个流量包出现了异常现象，我们可以修复一下这个流量包。

这里我们用到一个在线工具：<http://f00l.de/hacking/pcapfix.php>

这个工具可以帮助我们快速地将其流量包修复为 pcap 包。

我们对其进行在线修复。

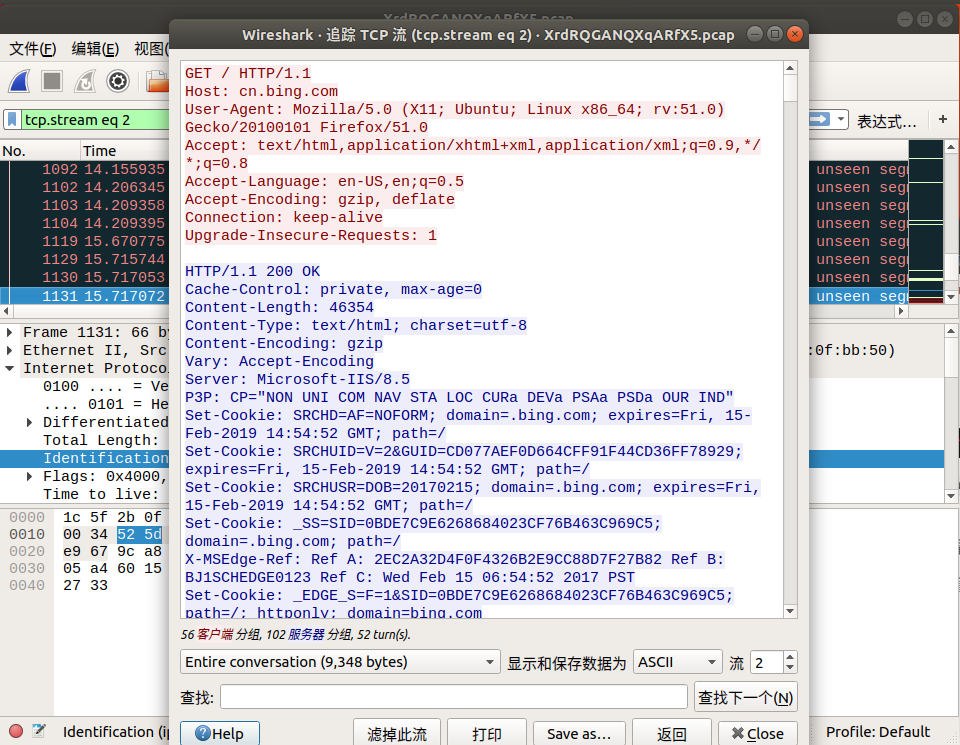


修复完毕后点击 Get your repaired PCAP-file here. 即可下载流量包，然后我们用 wireshark 打开。

既然还是要找 flag ，我们可以先看看这个流量包。

**第三步，追踪 TCP 流**

我们追踪一下 TCP 流，看看有没有什么突破？



我们通过追踪 TCP 流，可以看到一些版本信息， cookie 等等，我们还是发现了一些很有意思的东西。

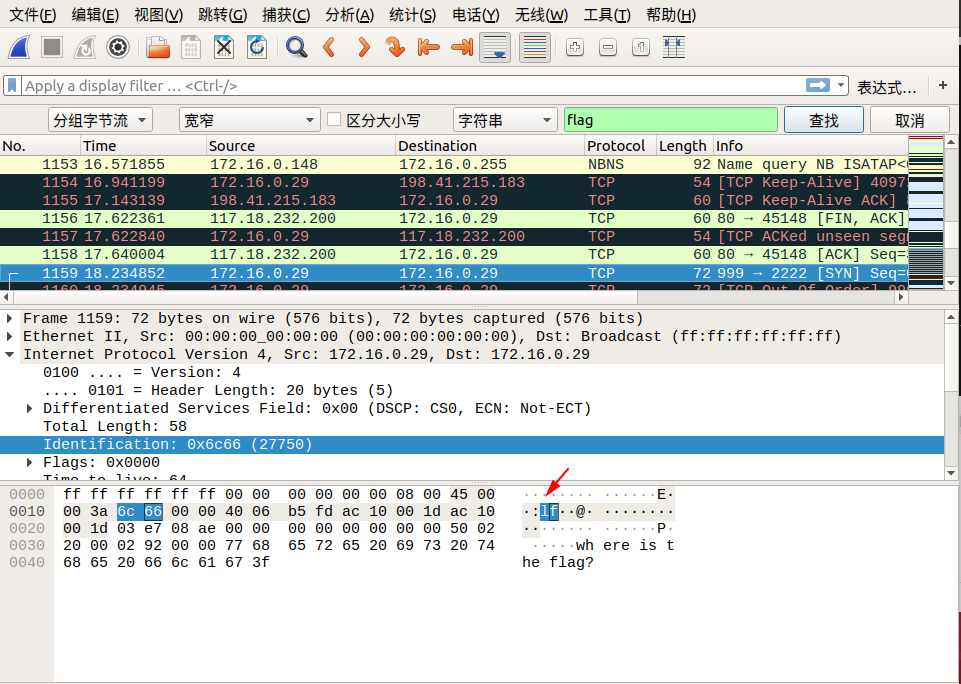
从 tcp.stream eq 29 到 tcp.stream eq 41 只显示了 where is the flag? 这个字样，难道这是出题人在告诉我们 flag 在这里嘛？

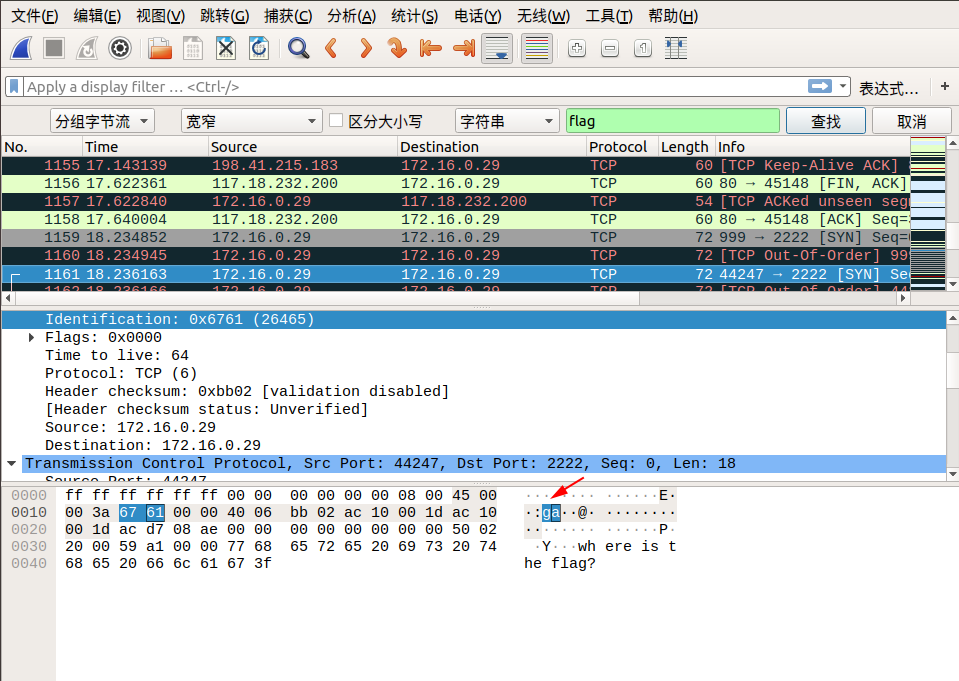
**第四步，查找分组字节流**

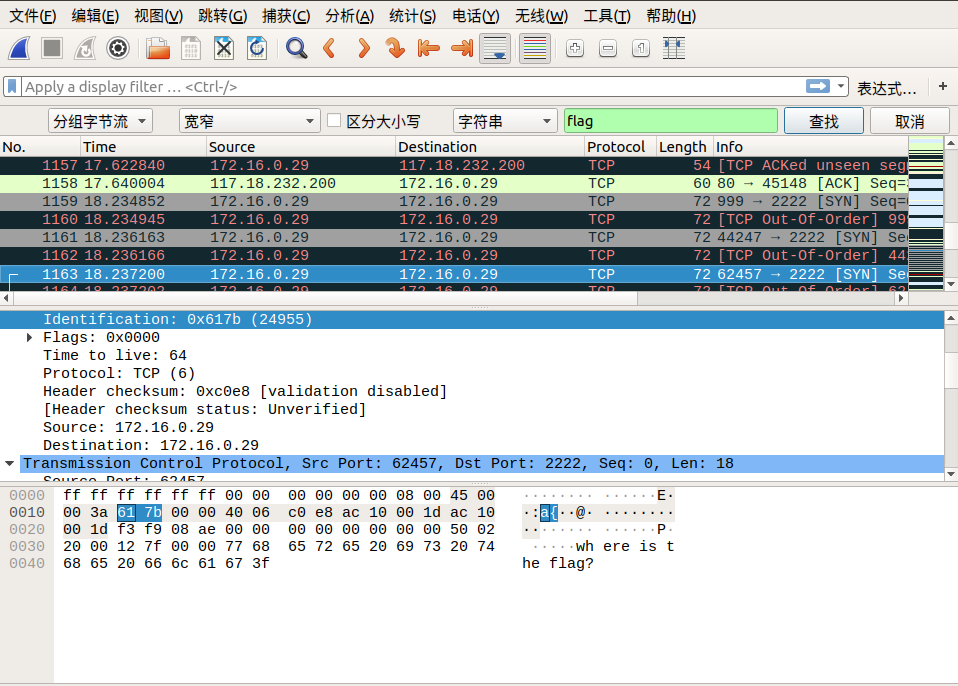
我们追踪到 tcp.stream eq 29 的时候，在 Identification 信息中看到了 flag 中的 lf 字样，我们可以继续追踪下一个流，在 tcp.stream eq 30 的 Identification 信息中看到了 flag 中的 ga 字样，我们发现将两个包中 Identification 信息对应的字段从右至左组合，恰好就是 flag ！于是我们可以大胆地猜测， flag 肯定是藏在这里面。

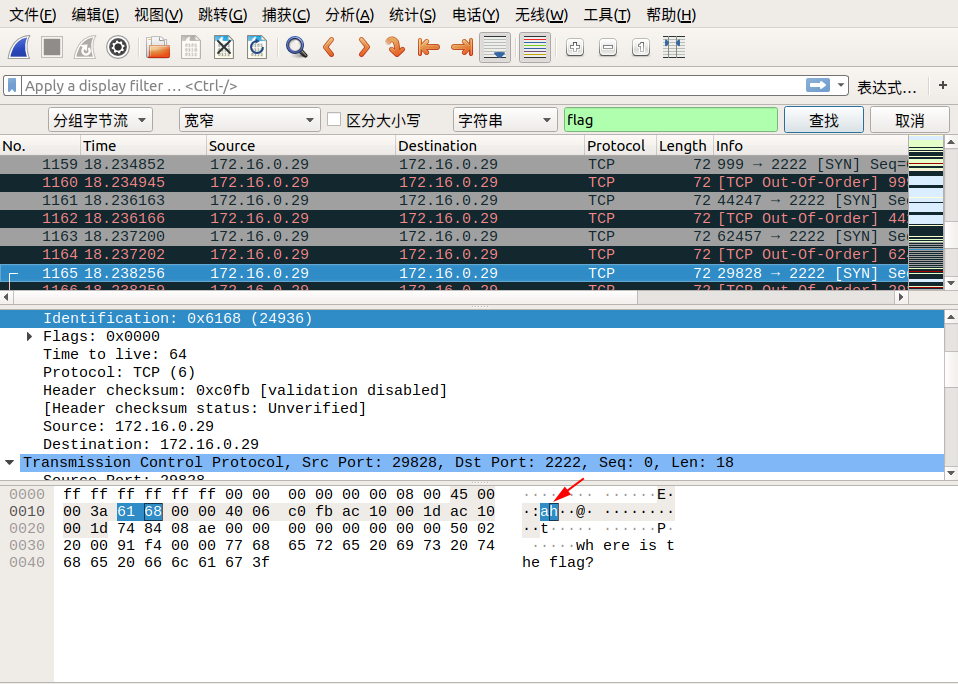
我们直接通过搜索 -> 字符串搜索 -> 分组字节流 -> 搜索关键字 flag 即可，按照同样的方式连接后面相连数据包的 Identification 信息对应的字段，即可找到最终的 flag！

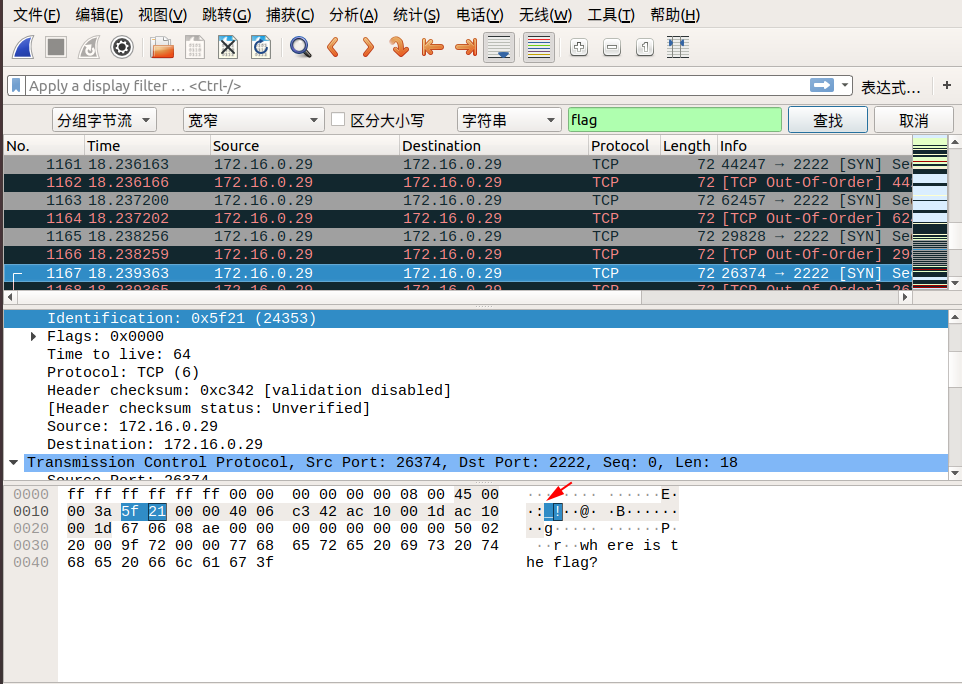
下面是搜索的截图：

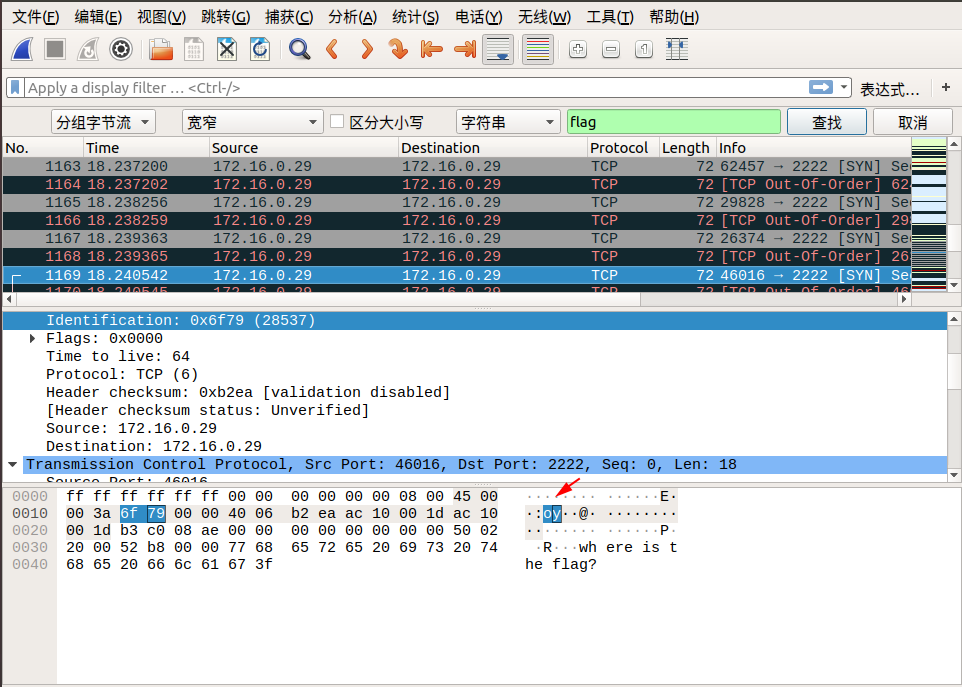


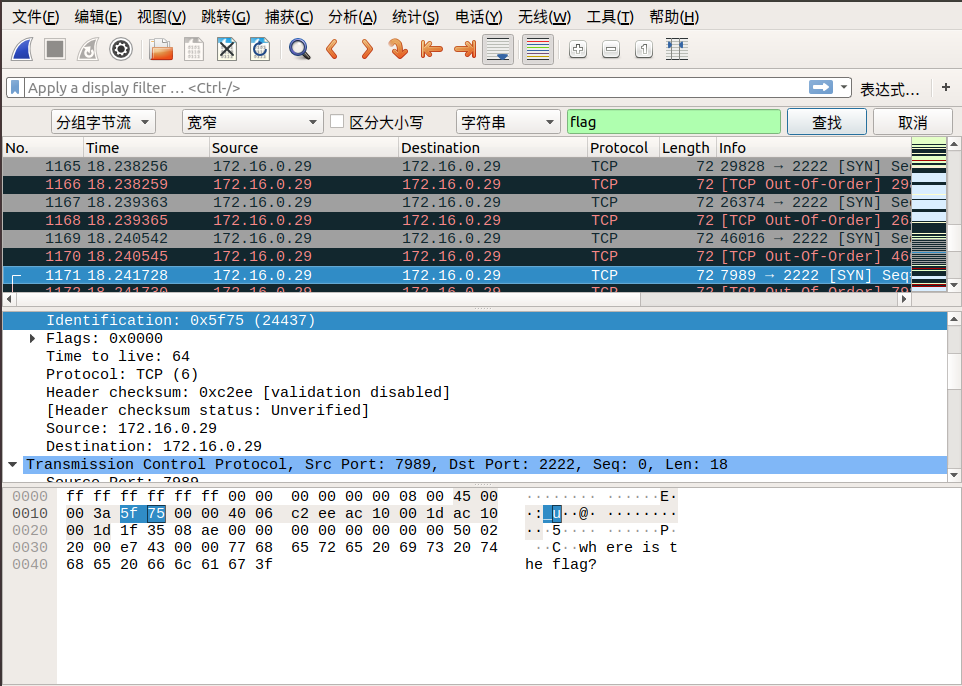




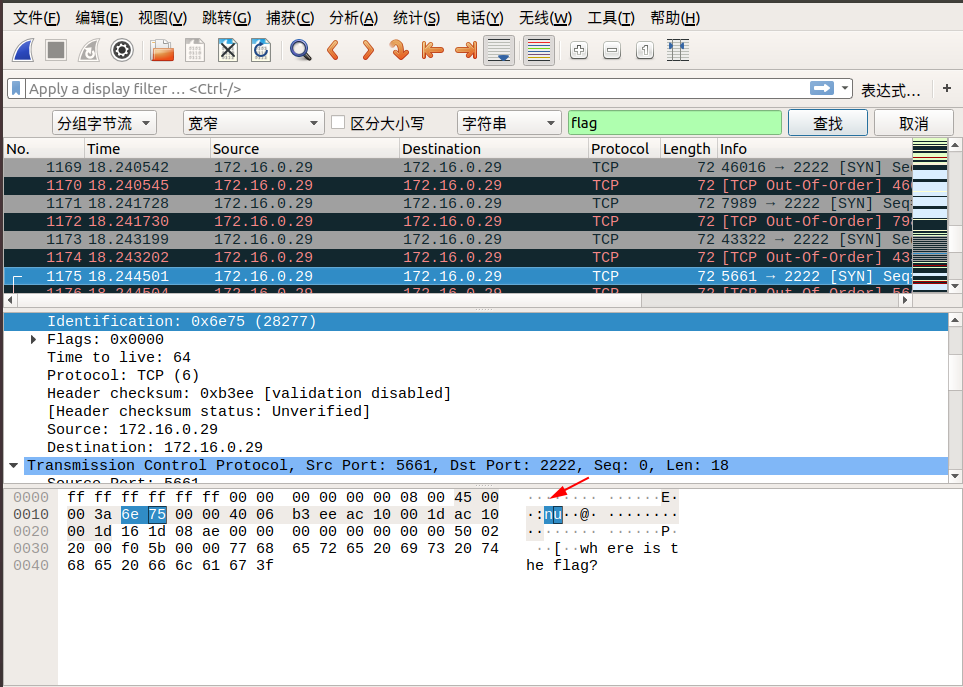


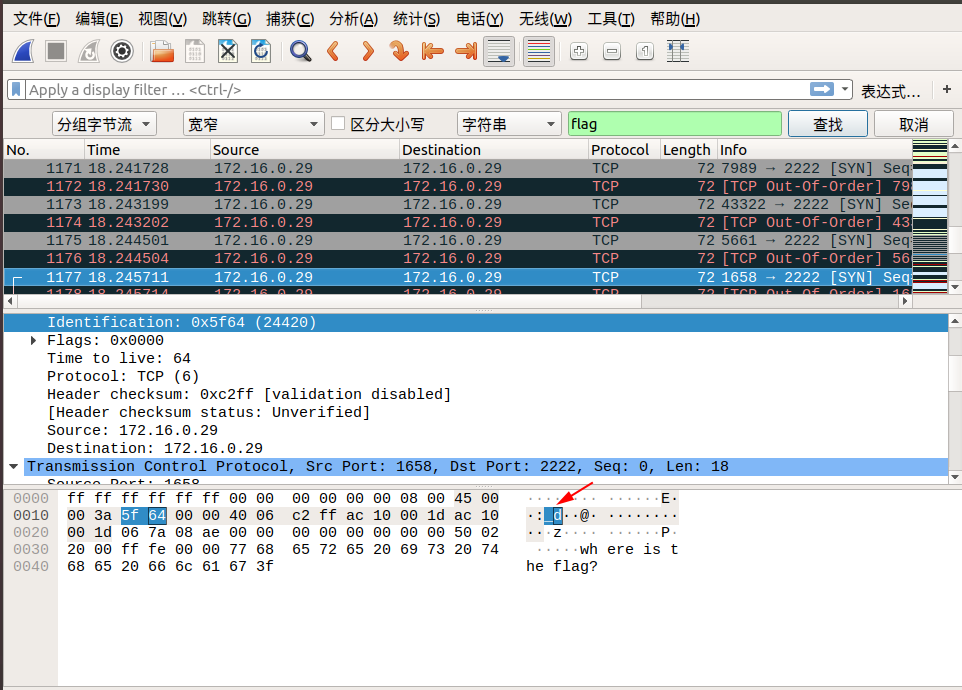


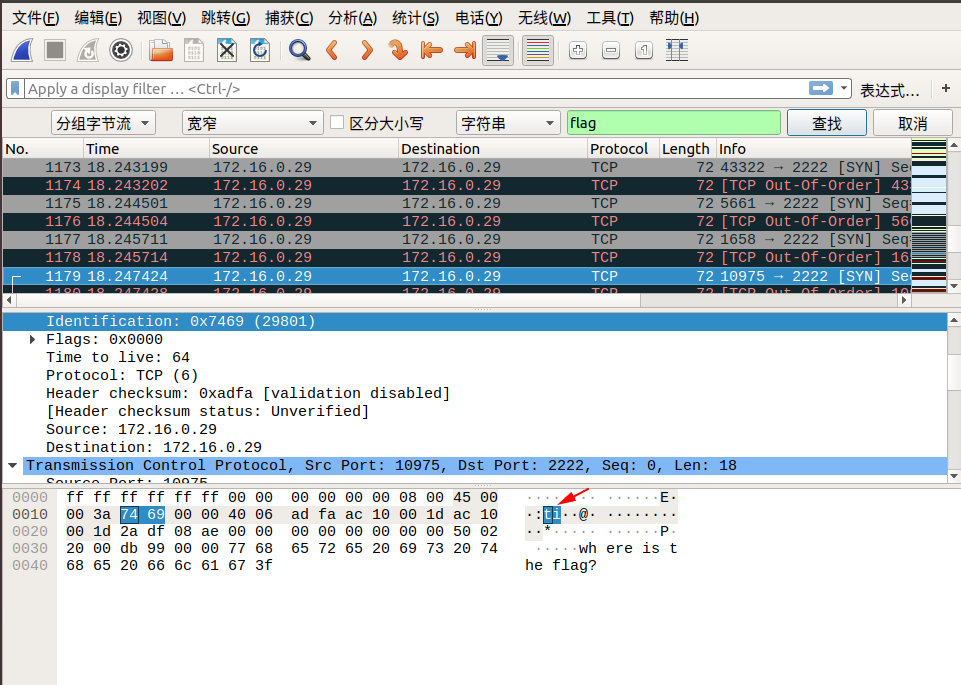


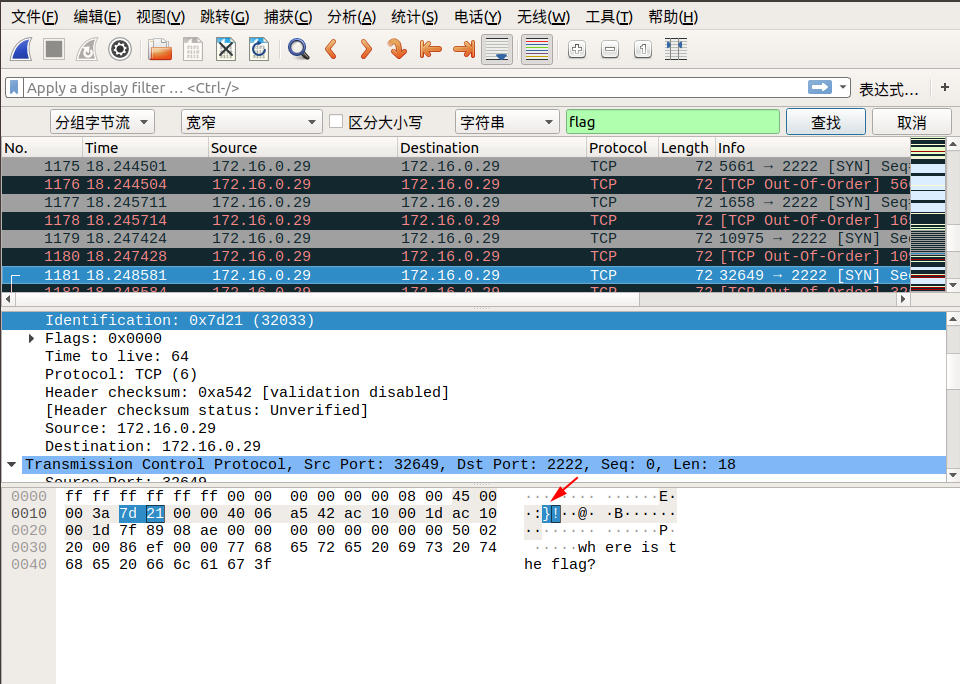












所以最终的 flag 为：**flag{aha!\_you\_found\_it!}**

1. **协议分析**

# 协议分析概述

网络协议为计算机网络中进行数据交换而建立的规则、标准或约定的集合。例如，网络中一个微机用户和一个大型主机的操作员进行通信，由于这两个数据终端所用字符集不同，因此操作员所输入的命令彼此不认识。为了能进行通信，规定每个终端都要将各自字符集中的字符先变换为标准字符集的字符后，才进入网络传送，到达目的终端之后，再变换为该终端字符集的字符。当然，对于不相容终端，除了需变换字符集字符外还需转换其他特性，如显示格式、行长、行数、屏幕滚动方式等也需作相应的变换。

相应的，我们在协议分析这一章节中，将会从以下几个方面来介绍这一部分的知识：

* Wireshark 常用功能的介绍
* HTTP 协议分析
* HTTPS 协议分析
* FTP 协议分析
* DNS 协议分析
* WIFI 协议分析
* USB 协议分析

1. **Wireshark**

## Wireshark 常用功能介绍

### 显示过滤器

显示过滤器可以用很多不同的参数来作为匹配标准，比如 IP 地址、协议、端口号、某些协议头部的参数。此外，用户也用一些条件工具和串联运算符创建出更加复杂的表达式。用户可以将不同的表达式组合起来，让软件显示的数据包范围更加精确。在数据包列表面板中显示的所有数据包都可以用数据包中包含的字段进行过滤。

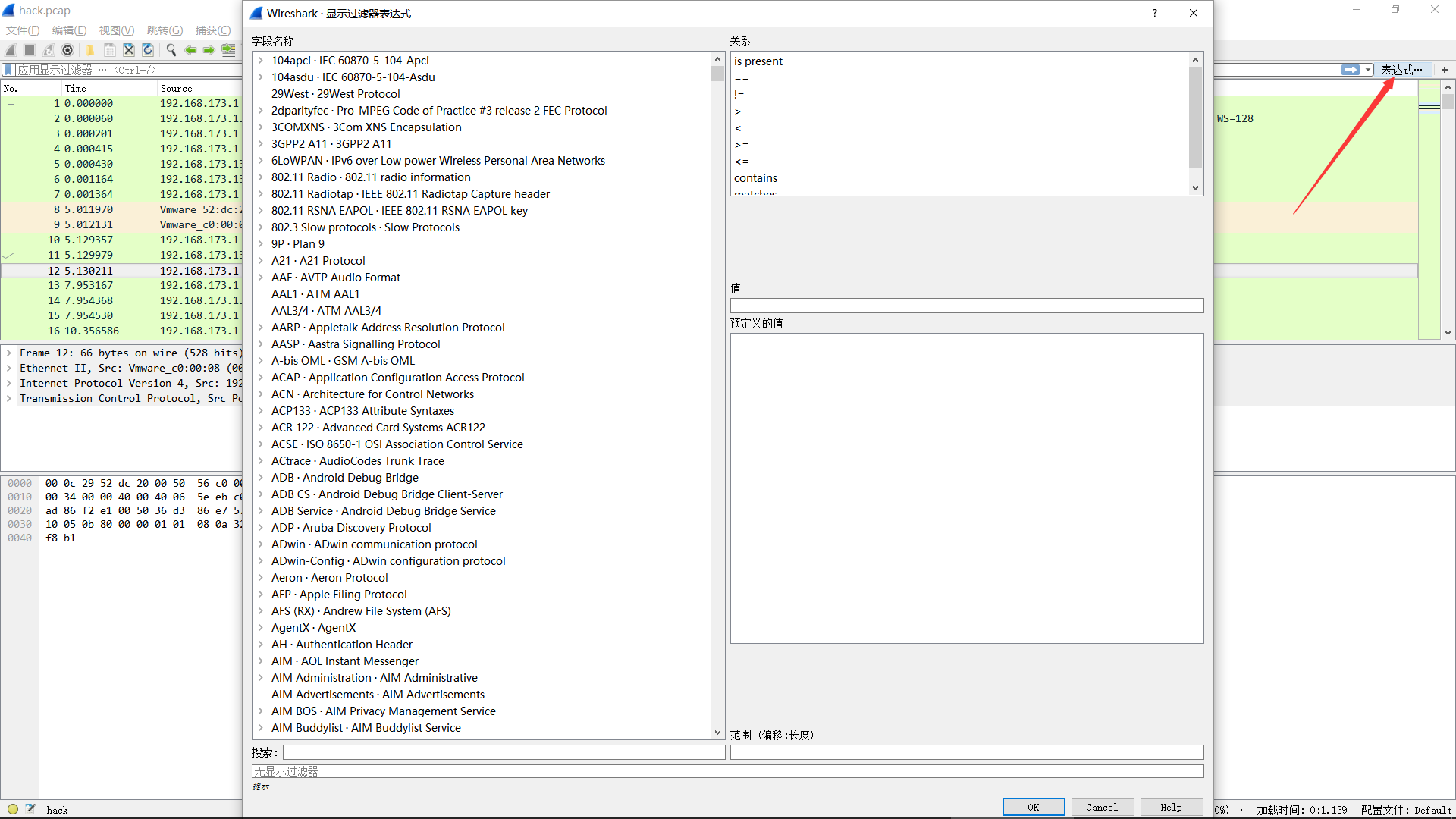
[not] Expression [and|or] [not] Expression

经常要用到各种运算符

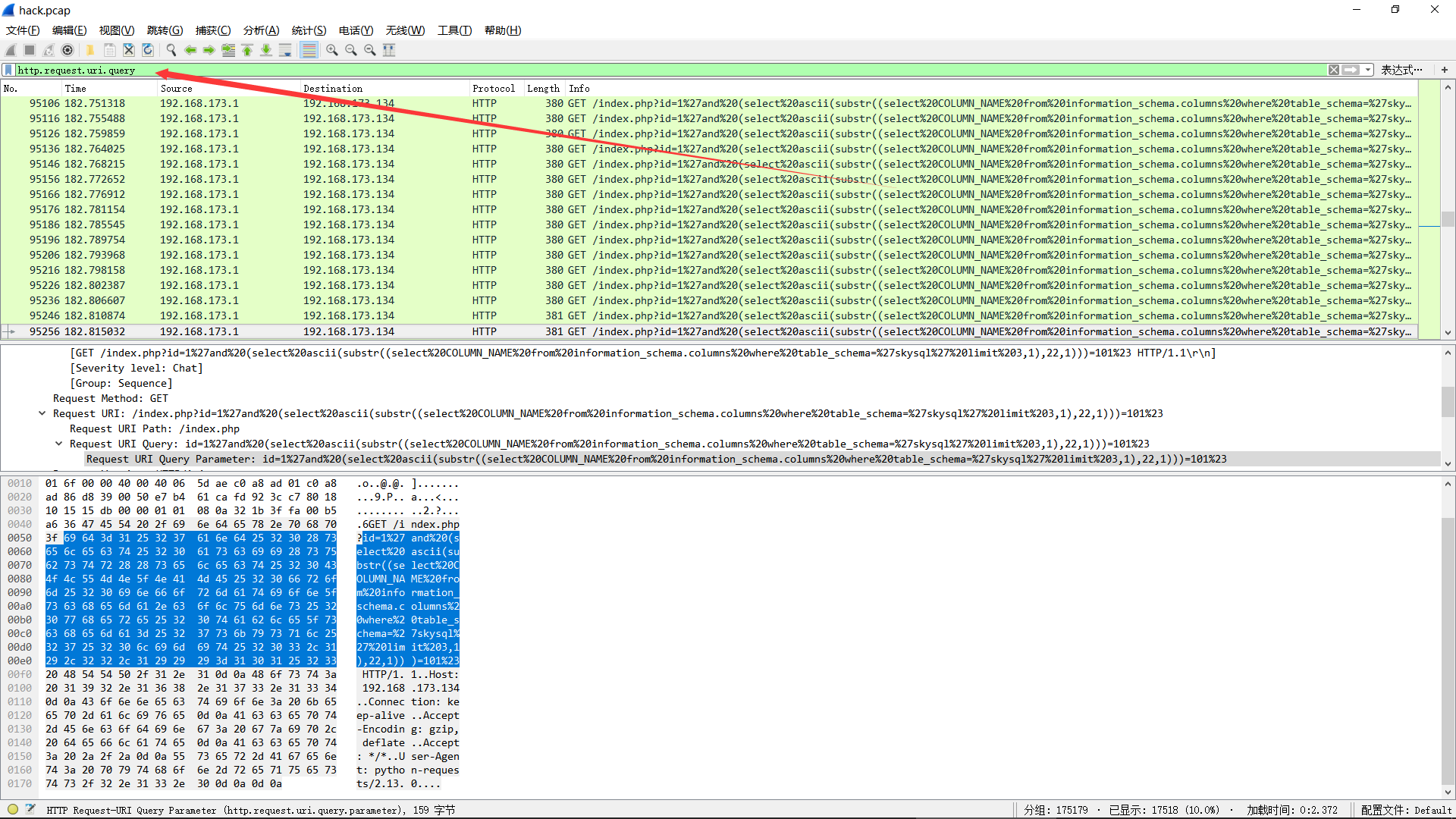
|  |  |
| --- | --- |
| 运算符 | 说明 |
| == | 等于 |
| != | 不等于 |
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |
| 与 | and ,&& |
| 或 | or,|| |
| 非 | !,not |

#### **配置方法**

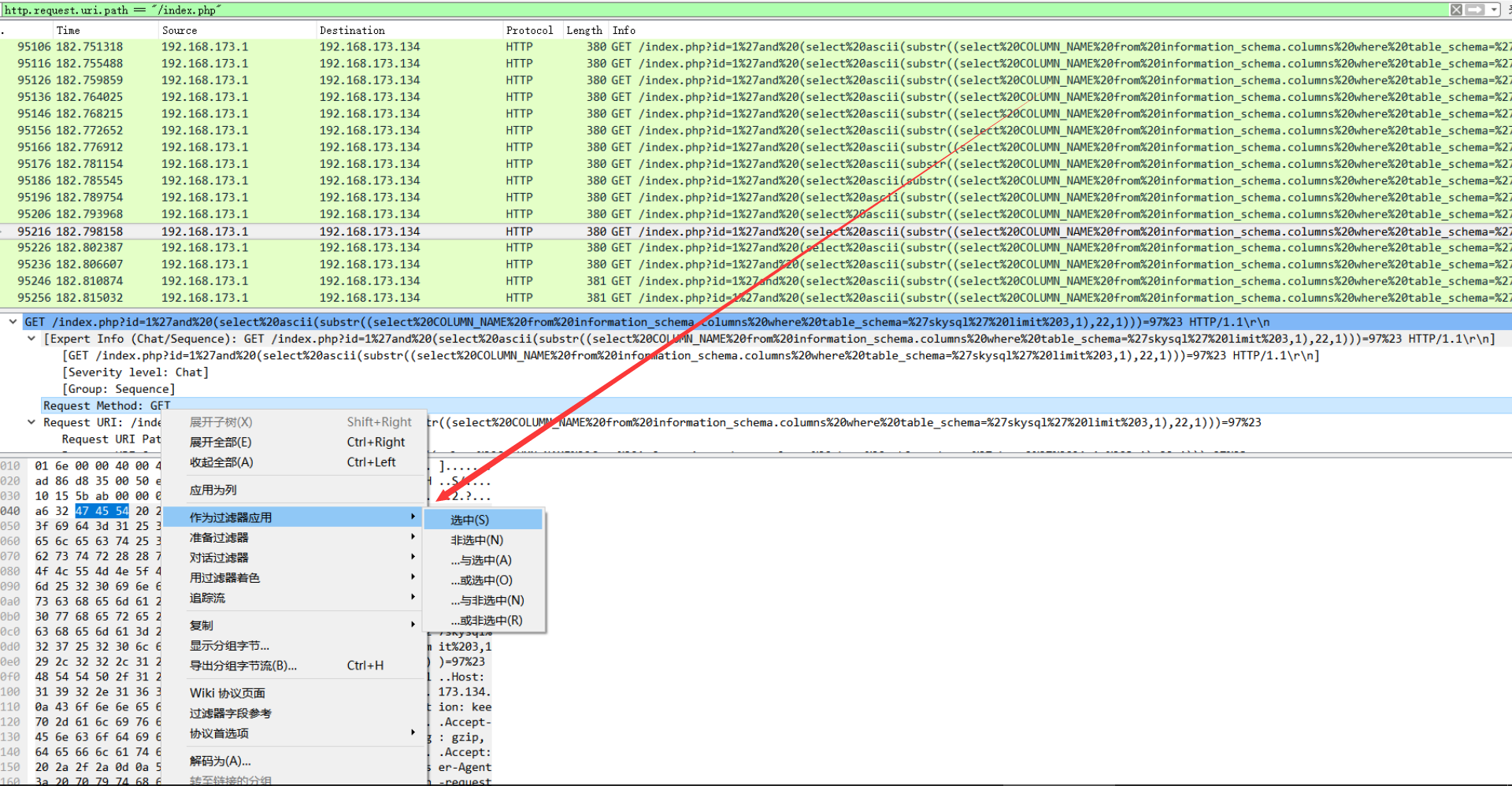
1. 借助于过滤器窗口



1. 借助于工具条的输入栏



1. 将数据包某个属性值指定为过滤条件



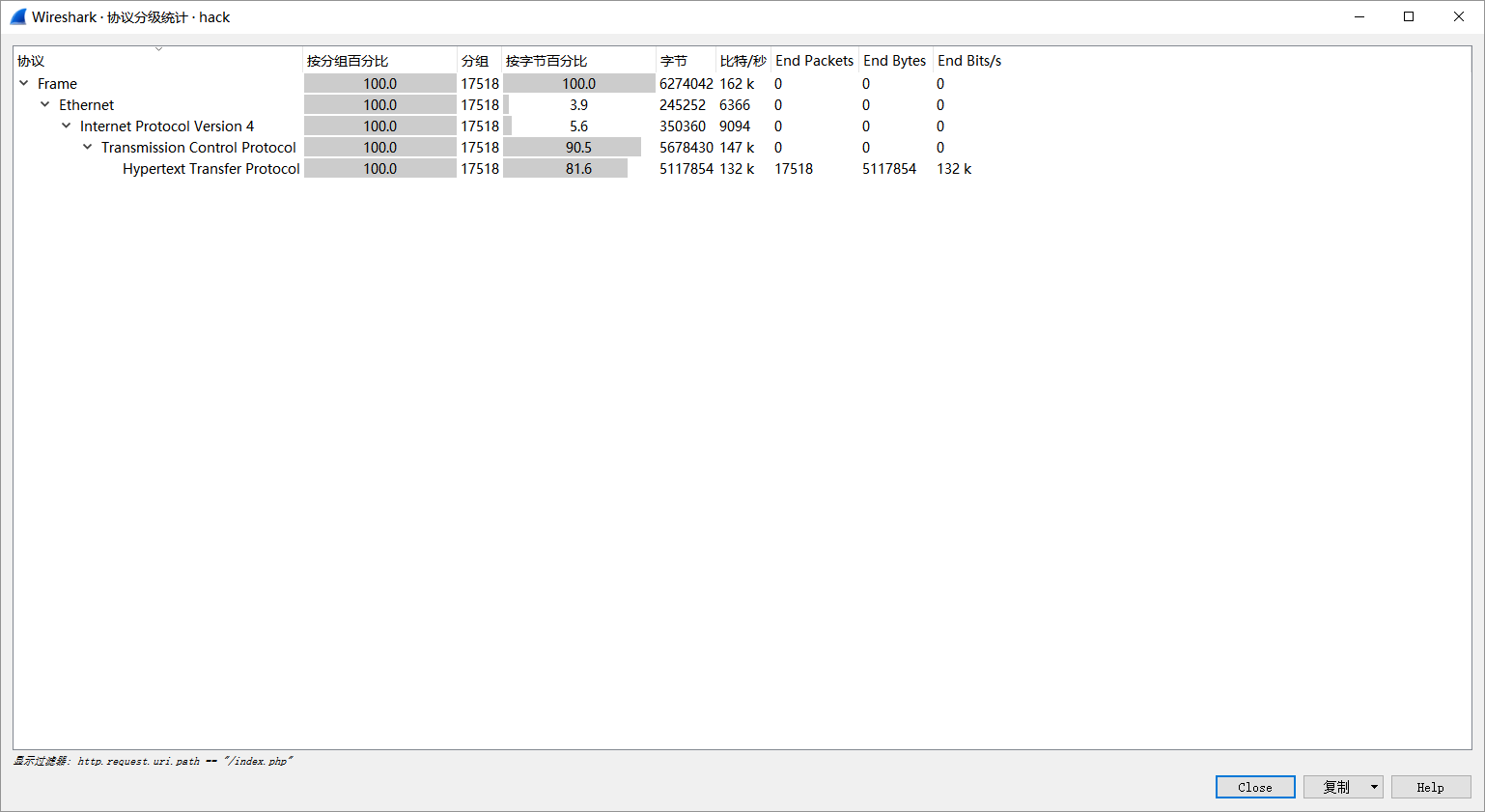
**Note**

​ 复杂的过滤命令可以直接通过第三种方式得到过滤语法

### 信息统计

#### **Protocol History(协议分级)**

这个窗口现实的是捕捉文件包含的所有协议的树状分支



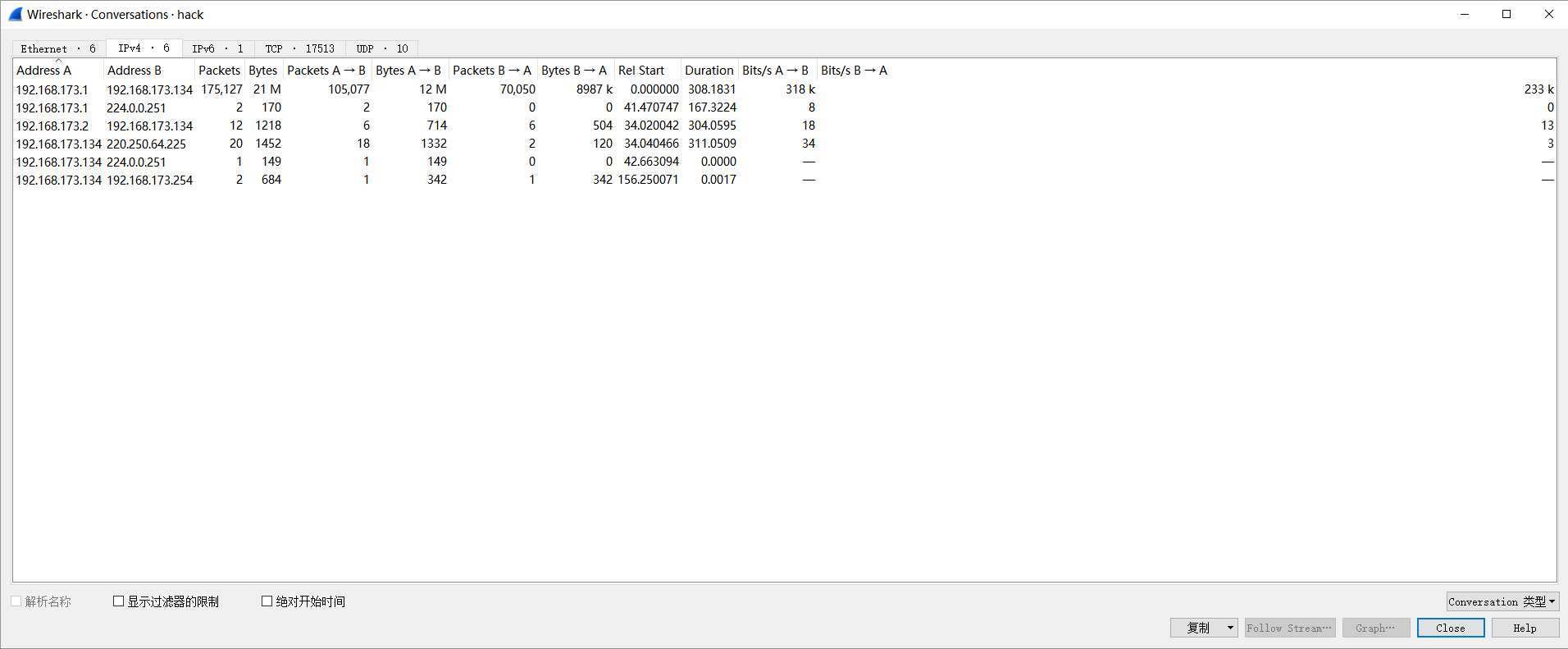
包含的字段

|  |  |
| --- | --- |
| 名称 | 含义 |
| Protocol | 协议名称 |
| % Packets | 含有该协议的包数目在捕捉文件所有包所占的比例 |
| Packets | 含有该协议的包的数目 |
| Bytes | 含有该协议的字节数 |
| Mbit/s | 抓包时间内的协议带宽 |
| End Packets | 该协议中的包的数目（作为文件中的最高协议层） |
| End Bytes | 该协议中的字节数（作为文件中的最高协议层） |
| End Mbit/s | 抓包时间内的协议带宽（作为文件中的最高协议层） |

这一功能可以为分析数据包的主要方向提供依据

#### **Conversation(对话)**

发生于一特定端点的 IP 间的所有流量.

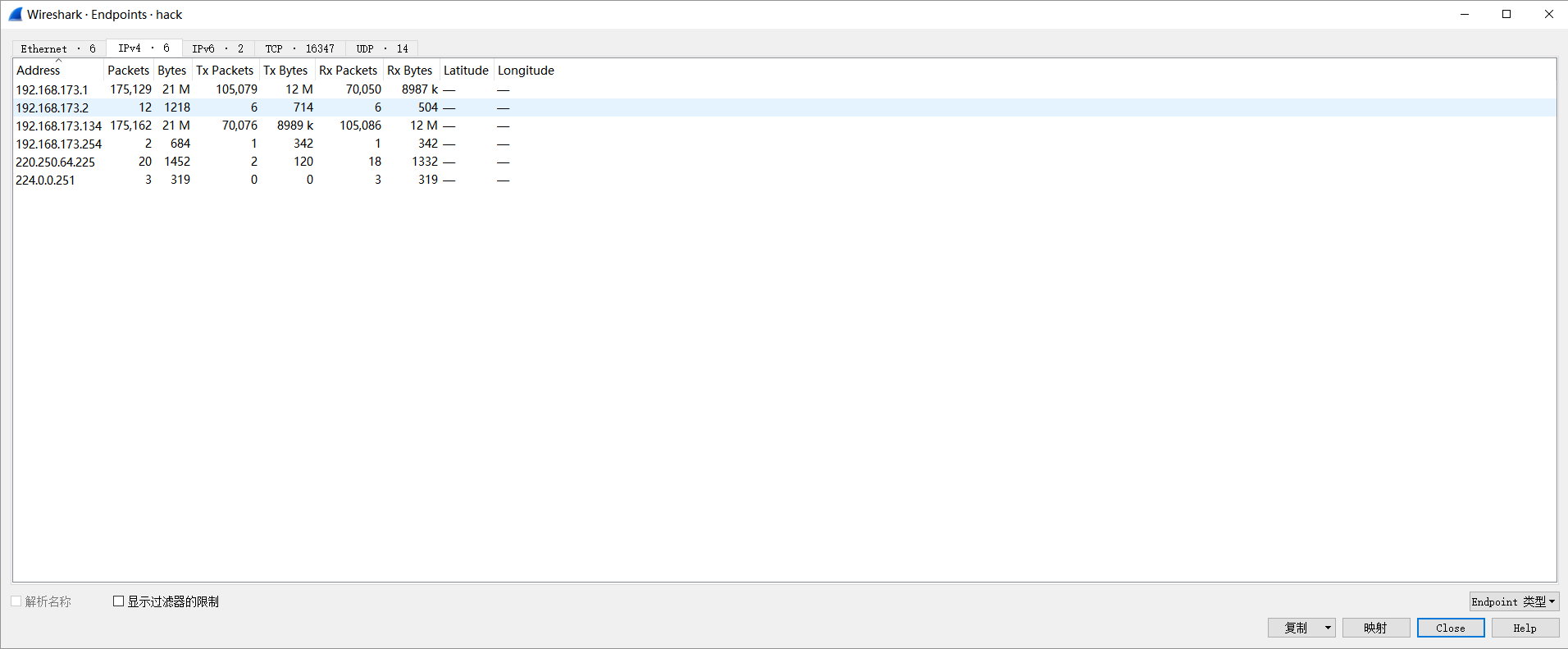


**Note**

​ - 查看收发大量数据流的 IP 地址。如果是你知道的服务器（你记得服务器的地址或地址范围），那问题就解决了；但也有可能只是某台设备正在扫描网络，或仅是一台产生过多数据的 PC。 ​ - 查看扫描模式（scan pattern）。这可能是一次正常的扫描，如 SNMP 软件发送 ping 报文以查找网络，但通常扫描都不是好事情

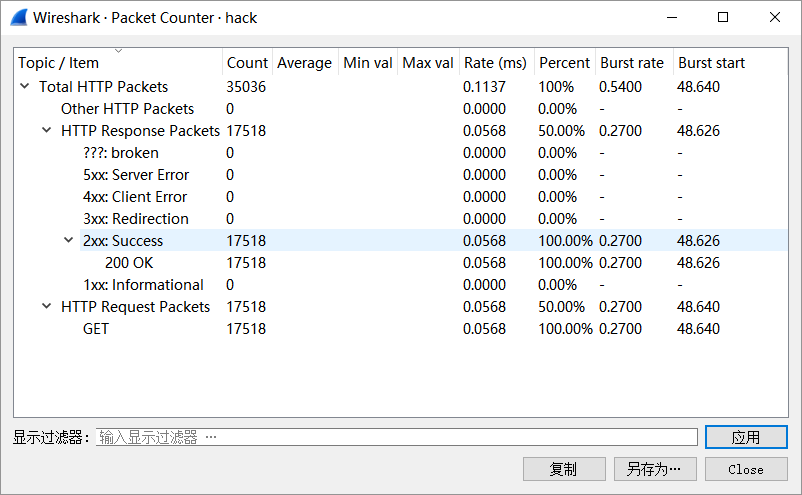
#### **EndPoints(端点)**

这一工具列出了 Wireshark 发现的所有 endpoints 上的统计信息



#### **HTTP**

* Packet Counter



**参考**

* <http://blog.jobbole.com/73482/>
* <http://www.vuln.cn/2103>

### 信息统计 进阶版

根据总体信息迅速把握流量包总体特征, 搞清楚 **做什么? TODO**

# 3. HTTP

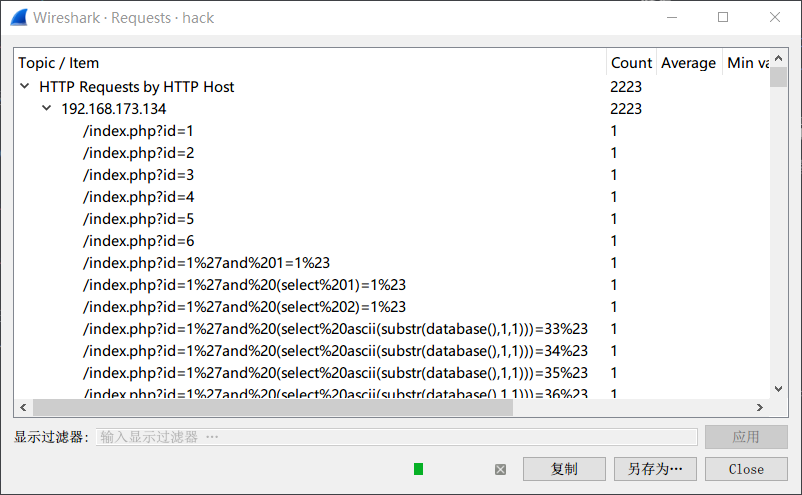
HTTP ( Hyper Text Transfer Protocol ，也称为超文本传输协议) 是一种用于分布式、协作式和超媒体信息系统的应用层协议。 HTTP 是万维网的数据通信的基础。

## 例题

题目：江苏省领航杯 - 2017：hack

总体观察可以得出:

* HTTP为主
* 192.168.173.134为主
* 不存在附件



从这张图, 基本可以判断初这是一个在sql注入-盲注时产生的流量包

到此为止, 基本可以判断 flag 的方向, 提取出所有的 url 后, 用python辅助即可得到 flag

* 提取 url: tshark -r hack.pcap -T fields -e http.request.full\_uri|tr -s '\n'|grep flag > log
* 得到盲注结果

import re

with open('log') as f:

tmp = f.read()

flag = ''

data = re.findall(r'=(\d\*)%23',tmp)

data = [int(i) for i in data]

for i,num in enumerate(data):

try:

if num > data[i+1]:

flag += chr(num)

except Exception:

pass

print flag

1. **HTTPS**

HTTPs = HTTP + SSL / TLS. 服务端和客户端的信息传输都会通过 TLS 进行加密，所以传输的数据都是加密后的数据

* [wireshark 分析 HTTPs](http://www.freebuf.com/articles/system/37900.html)

## 例题

题目：hack-dat-kiwi-ctf-2015:ssl-sniff-2

打开流量包发现是 SSL 加密过的数据, 导入题目提供的server.key.insecure, 即可解密

GET /key.html HTTP/1.1

Host: localhost

HTTP/1.1 200 OK

Date: Fri, 20 Nov 2015 14:16:24 GMT

Server: Apache/2.4.7 (Ubuntu)

Last-Modified: Fri, 20 Nov 2015 14:15:54 GMT

ETag: "1c-524f98378d4e1"

Accept-Ranges: bytes

Content-Length: 28

Content-Type: text/html

The key is 39u7v25n1jxkl123

1. **FTP**

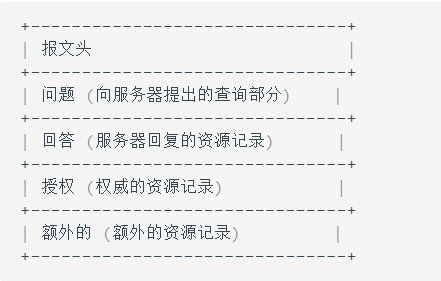
FTP ( File Transfer Protocol ，即文件传输协议) 是 TCP/IP 协议组中的协议之一。 FTP 协议包括两个组成部分，其一为 FTP 服务器，其二为 FTP 客户端。其中 FTP 服务器用来存储文件，用户可以使用 FTP 客户端通过 FTP 协议访问位于 FTP 服务器上的资源。在开发网站的时候，通常利用 FTP 协议把网页或程序传到 Web 服务器上。此外，由于 FTP 传输效率非常高，在网络上传输大的文件时，一般也采用该协议。

默认情况下 FTP 协议使用 TCP 端口中的 20 和 21 这两个端口，其中 20 用于传输数据， 21用于传输控制信息。但是，是否使用 20 作为传输数据的端口与 FTP 使用的传输模式有关，如果采用主动模式，那么数据传输端口就是 20 ；如果采用被动模式，则具体最终使用哪个端口要服务器端和客户端协商决定。

1. **DNS**

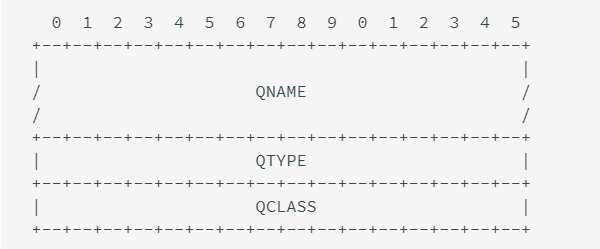
## 简介

DNS 通常为 UDP 协议, 报文格式



查询包只有头部和问题两个部分， DNS 收到查询包后，根据查询到的信息追加回答信息、授权机构、额外资源记录，并且修改了包头的相关标识再返回给客户端。

每个 question 部分



* QNAME ：为查询的域名，是可变长的，编码格式为：将域名用. 号划分为多个部分，每个部分前面加上一个字节表示该部分的长度，最后加一个 0 字节表示结束
* QTYPE ：占 16 位，表示查询类型，共有 16 种，常用值有：1 ( A 记录，请求主机 IP 地址)、2 ( NS ，请求授权 DNS 服务器)、5 ( CNAME 别名查询)

## 例题

题目：BSides San Francisco CTF 2017 ： dnscap.pcap

我们通过 wireshark 打开发现全部为 DNS 协议, 查询名为大量字符串([\w\.]+)\.skullseclabs\.org

我们通过 tshark -r dnscap.pcap -T fields -e dns.qry.name > hex提取后，利用 python 转码：

import re

find = ""

with open('hex','rb') as f:

for i in f:

text = re.findall(r'([\w\.]+)\.skull',i)

if text:

find += text[0].replace('.','')

print find

我们发现了几条关键信息：

Welcome to dnscap! The flag is below, have fun!!

Welcome to dnscap! The flag is below, have fun!!

!command (sirvimes)

...

IHDR

gAMA

bKGD

pHYs

IHDR

gAMA

bKGD

pHYs

tIME

IDATx

...

2017-02-01T21:04:00-08:00

IEND

console (sirvimes)

console (sirvimes)

Good luck! That was dnscat2 traffic on a flaky connection with lots of re-transmits. Seriously,

Good luck! That was dnscat2 traffic on a flaky connection with lots of re-transmits. Seriously, d[

good luck. :)+

flag 确实包含在其中, 但是有大量重复信息, 一是应为question 。在 dns 协议中查询和反馈时都会用到，-Y "ip.src == 192.168.43.91"进行过滤后发现还是有不少重复部分。

%2A}

%2A}

%2A}q

%2A}x

%2A}

IHDR

gAMA

bKGD

pHYs

tIME

IDATx

HBBH

CxRH!

C1%t

ceyF

i4ZI32

rP@1

ceyF

i4ZI32

rP@1

ceyF

i4ZI32

rP@1

ceyF

i4ZI32

rP@1

根据发现的 dnscat 找到 <https://github.com/iagox86/dnscat2/blob/master/doc/protocol.md> 这里介绍了 dnscat 协议的相关信息, 这是一种通过 DNS 传递数据的变种协议, 题目文件中应该未使用加密, 所以直接看这里的数据块信息

MESSAGE\_TYPE\_MSG: [0x01]

(uint16\_t) packet\_id

(uint8\_t) message\_type [0x01]

(uint16\_t) session\_id

(uint16\_t) seq

(uint16\_t) ack

(byte[]) data

在qry.name中去除其余字段, 只留下 data 快, 从而合并数据, 再从 16 进制中检索89504e.....6082提取png, 得到 flag 。

import re

find = []

with open('hex','rb') as f:

for i in f:

text = re.findall(r'([\w\.]+)\.skull',i)

if text:

tmp = text[0].replace('.','')

find.append(tmp[18:])

last = []

for i in find:

if i not in last:

last.append(i)

print ''.join(last)

flag



## 相关题目

* [IceCTF-2016:Search](https://mrpnkt.github.io/2016/icectf-2016-search/)
* [EIS-2017:DNS 101](https://github.com/susers/Writeups/blob/master/2017/EIS/Misc/DNS 101/Write-up.md)

1. WIFI

802.11 是现今无线局域网通用的标准, 常见认证方式

* 不启用安全‍‍
* WEP‍‍
* WPA/WPA2-PSK（预共享密钥）‍‍
* PA/WPA2 802.1X （radius 认证）

## WPA-PSK

认证大致过程如下图



其中四次握手过程



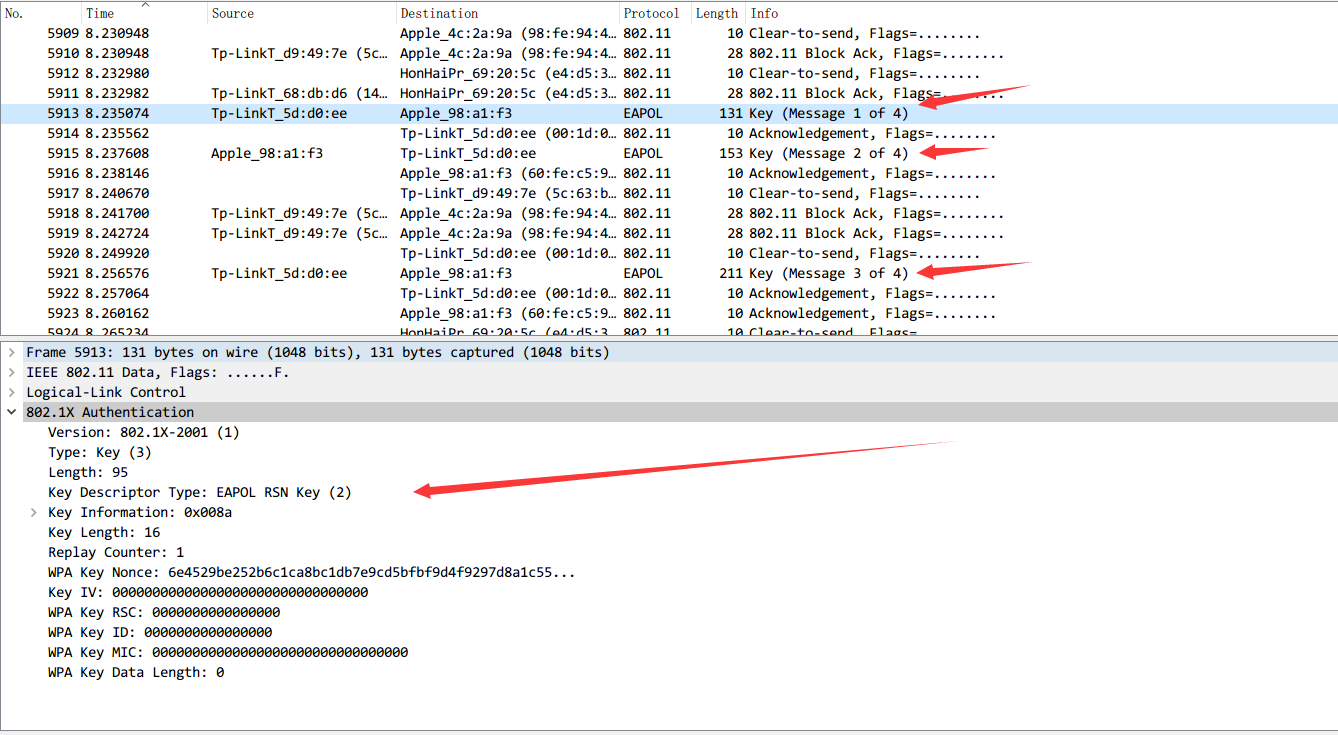
1. 4 次握手开始于验证器 (AP)，它产生一个随机的值(ANonce) 发送给请求者
2. 请求者也产生了它自己的随机 SNonce，然后用这两个 Nonces 以及 PMK 生成了 PTK。请求者回复消息 2 给验证器, 还有一个 MIC（message integrity code，消息验证码）作为 PMK 的验证
3. 它先要验证请求者在消息 2 中发来的 MIC 等信息，验证成功后，如果需要就生成 GTK。然后发送消息 3
4. 请求者收到消息 3，验证 MIC，安装密钥，发送消息 4，一个确认信息。验证器收到消息 4，验证 MIC，安装相同的密钥

## 例题

实验吧： shipin.cap

从大量的Deauth 攻击基本可以判断是一个破解 wifi 时的流量攻击

同时也成功发现了握手包信息



接下来跑密码

* linux ： aircrack 套件
* windows ： wifipr ，速度比 esaw 快， GTX850 能将近 10w\s :)

得到密码88888888在 wireshark 中Edit -> Preferences -> Protocols -> IEEE802.11 -> Edit以key:SSID形式填入即可解密 wifi 包看到明文流量

KCARCK 相关: <https://www.krackattacks.com/>

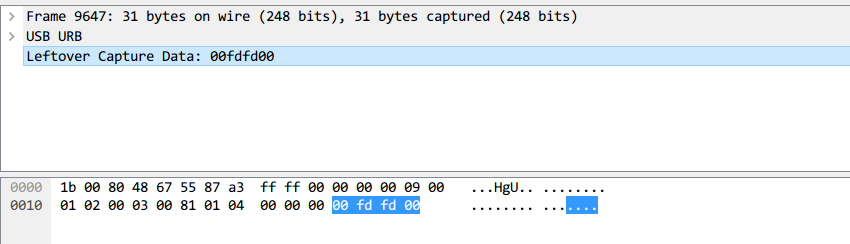
# USB

## 简介

**USB 详述**: <https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf>

* 鼠标协议

鼠标移动时表现为连续性，与键盘击键的离散性不一样，不过实际上鼠标动作所产生的数据包也是离散的，毕竟计算机表现的连续性信息都是由大量离散信息构成的

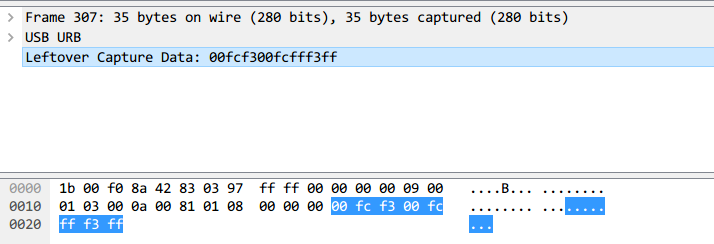


每一个数据包的数据区有四个字节，第一个字节代表按键，当取 0x00 时，代表没有按键、为 0x01 时，代表按左键，为 0x02 时，代表当前按键为右键。第二个字节可以看成是一个 signed byte 类型，其最高位为符号位，当这个值为正时，代表鼠标水平右移多少像素，为负时，代表水平左移多少像素。第三个字节与第二字节类似，代表垂直上下移动的偏移。

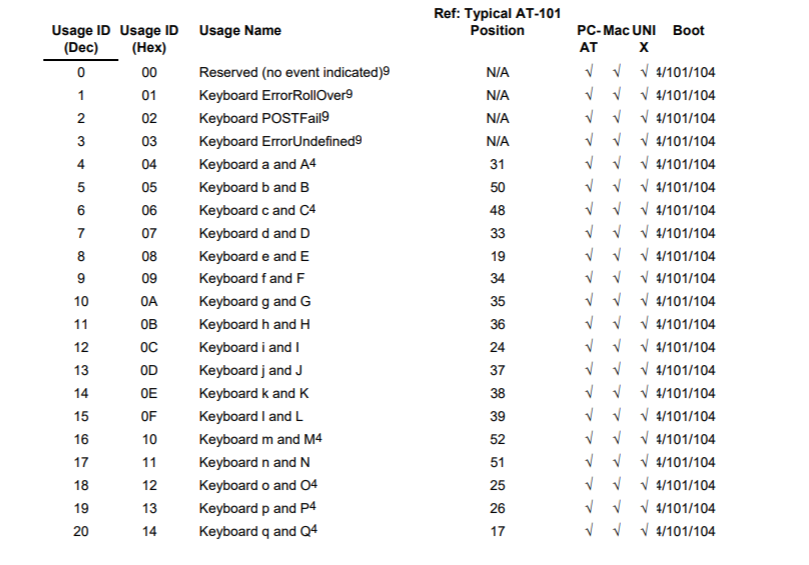
得到这些点的信息后, 即可恢复出鼠标移动轨迹

* Tools
* [UsbMiceDataHacker](https://github.com/WangYihang/UsbMiceDataHacker)
* 键盘协议

键盘数据包的数据长度为 8 个字节，击键信息集中在第 3 个字节



根据 data 值与具体键位的对应关系



可从数据包恢复出键盘的案件信息

* Tools
* [UsbKeyboardDataHacker](https://github.com/WangYihang/UsbKeyboardDataHacker)

**参考**

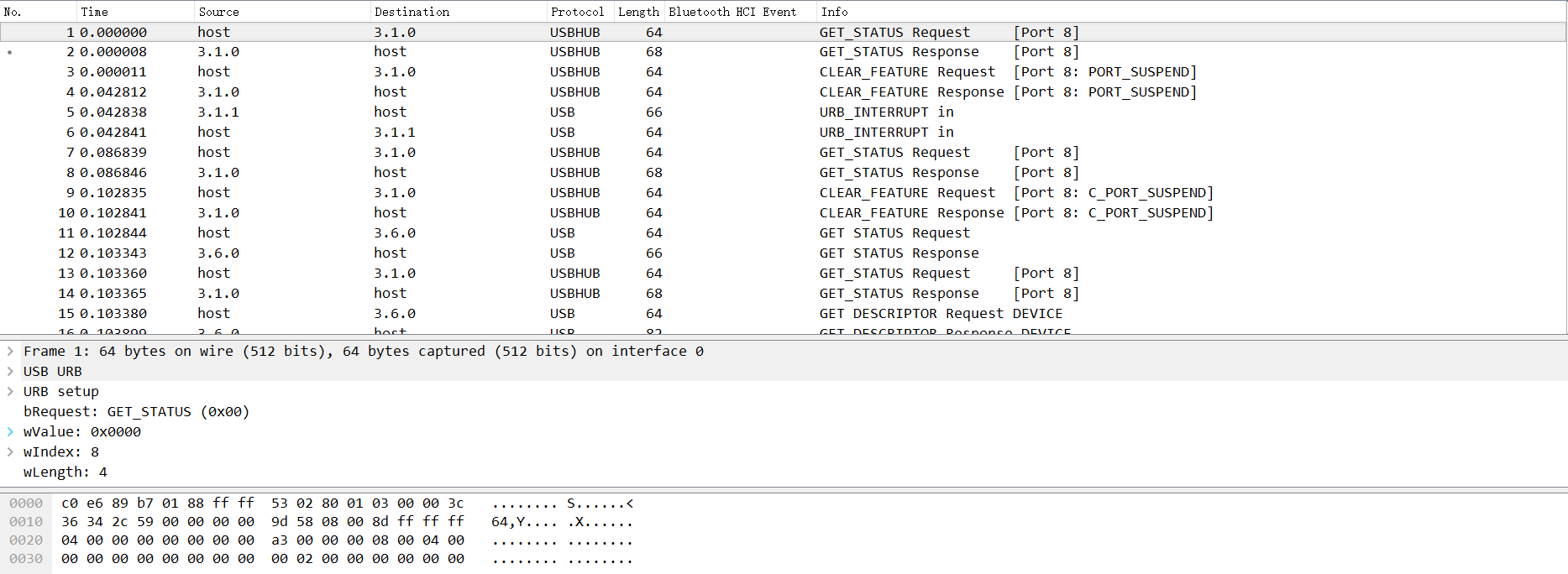
* <https://www.anquanke.com/post/id/85218>

## 例题

Xman三期夏令营排位赛练习题：AutoKey

WP：<https://www.cnblogs.com/ECJTUACM-873284962/p/9473808.html>

问题描述：



这道题是Xman 三期夏令营选拔赛出的一道题，我们如何对其进行分析？

### 流量包是如何捕获的？

首先我们从上面的数据包分析可以知道，这是个 USB 的流量包，我们可以先尝试分析一下 USB的数据包是如何捕获的。

在开始前，我们先介绍一些 USB 的基础知识。 USB 有不同的规格，以下是使用 USB 的三种方式：

l USB UART

l USB HID

l USB Memory

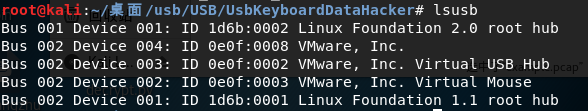
UART 或者 Universal Asynchronous Receiver/Transmitter 。这种方式下，设备只是简单的将 USB 用于接受和发射数据，除此之外就再没有其他通讯功能了。

HID 是人性化的接口。这一类通讯适用于交互式，有这种功能的设备有：键盘，鼠标，游戏手柄和数字显示设备。

最后是 USB Memory ，或者说是数据存储。 External HDD ， thumb drive/flash drive 等都是这一类的。

其中使用的最广的不是 USB HID 就是 USB Memory 了。

每一个 USB 设备（尤其是 HID 或者 Memory ）都有一个供应商 ID（Vendor ID） 和产品识别码（Product Id） 。 Vendor ID 是用来标记哪个厂商生产了这个 USB 设备。 Product ID 用来标记不同的产品，他并不是一个特殊的数字，当然最好不同。如下图：



上图是我在虚拟机环境下连接在我电脑上的 USB 设备列表，通过 lsusb 查看命令。

例如说，我在 VMware 下有一个无线鼠标。它是属于 HID 设备。这个设备正常的运行，并且通过lsusb 这个命令查看所有 USB 设备，现在大家能找出哪一条是这个鼠标吗？？没有错，就是第四个，就是下面这条：

Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse

其中，ID 0e0f:0003 就是 Vendor-Product ID 对， Vendor ID 的值是 0e0f ，并且 Product ID 的值是 0003 。 Bus 002 Device 002 代表 usb 设备正常连接，这点需要记下来。

我们用 root 权限运行 Wireshark 捕获 USB 数据流。但是通常来说我们不建议这么做。我们需要给用户足够的权限来获取 Linux 中的 usb 数据流。我们可以用 udev 来达到我们的目的。我们需要创建一个用户组 usbmon ，然后把我们的账户添加到这个组中。

addgroup usbmon

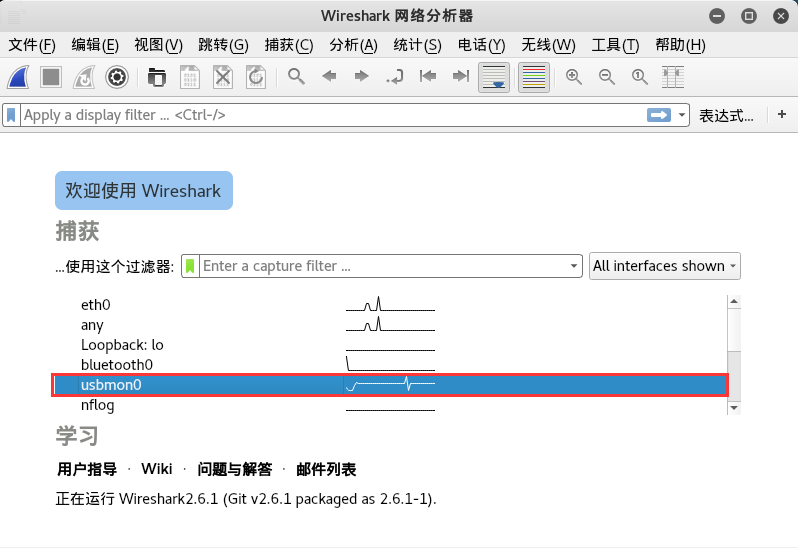
gpasswd -a $USER usbmon

echo 'SUBSYSTEM=="usbmon", GROUP="usbmon", MODE="640"' > /etc/udev/rules.d/99-usbmon.rules

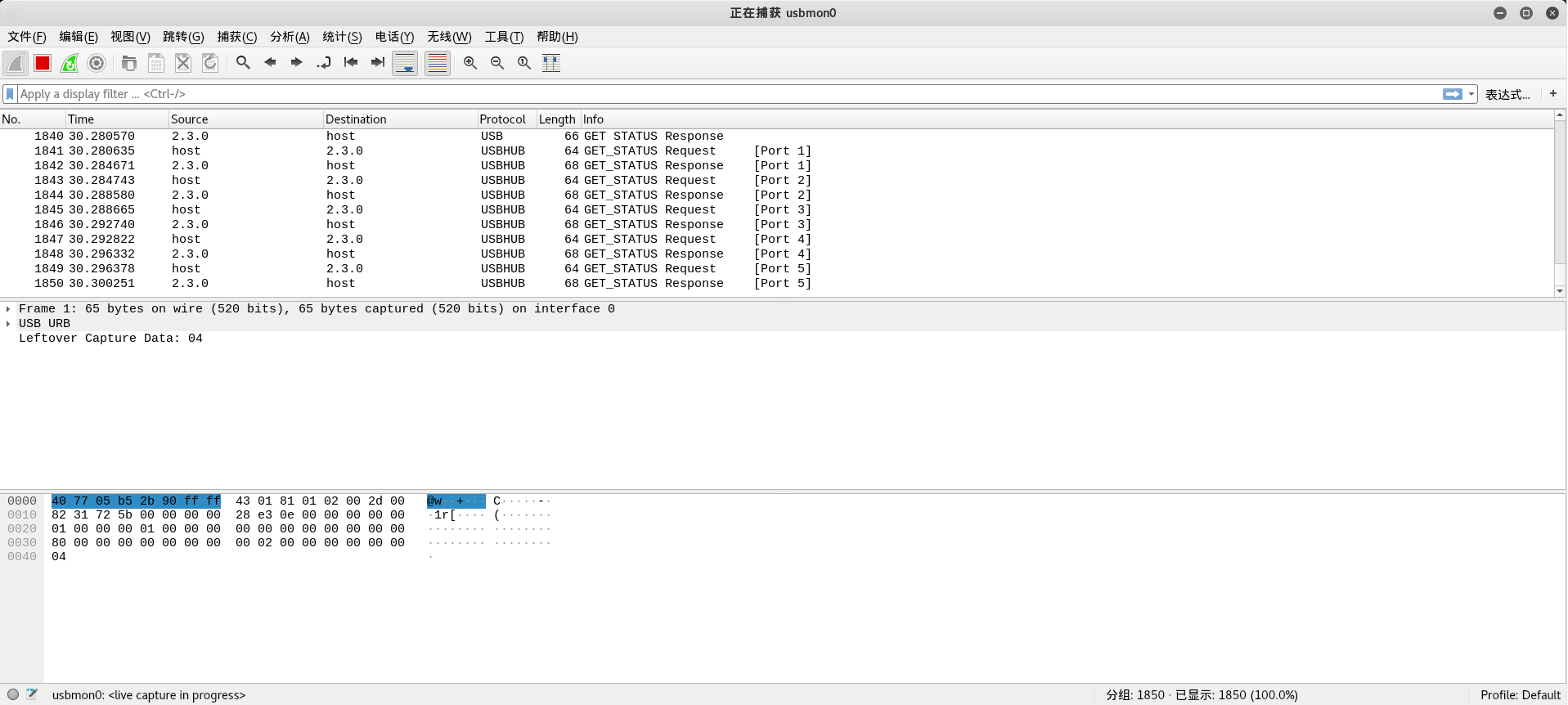
接下来，我们需要 usbmon 内核模块。如果该模块没有被加载，我们可以通过以下命令加载该模块：

modprobe usbmon

打开 wireshark ，你会看到 usbmonX 其中 X 代表数字。下图是我们本次的结果（我使用的是root）：



如果接口处于活跃状态或者有数据流经过的时候， wireshark 的界面就会把它以波形图的方式显示出来。那么，我们该选那个呢？没有错，就是我刚刚让大家记下来的，这个 X 的数字就是对应这 USB Bus 。在本文中是 usbmon0 。打开他就可以观察数据包了。



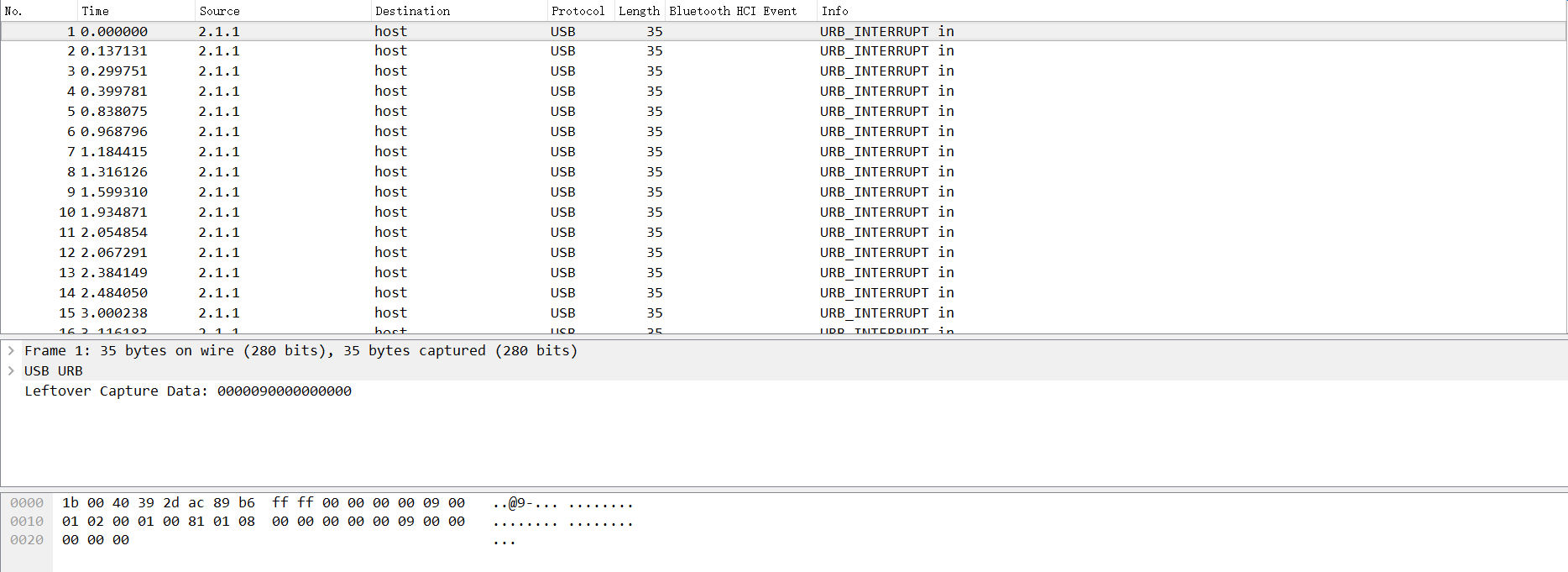
通过这些，我们可以了解到 usb 设备与主机之间的通信过程和工作原理，我们可以来对流量包进行分析了。

### 如何去分析一个 USB 流量包？

根据前面的知识铺垫，我们大致对 USB 流量包的抓取有了一个轮廓了，下面我们介绍一下如何分析一个 USB 流量包。

USB 协议的细节方面参考 wireshark 的 wiki ：<https://wiki.wireshark.org/USB>

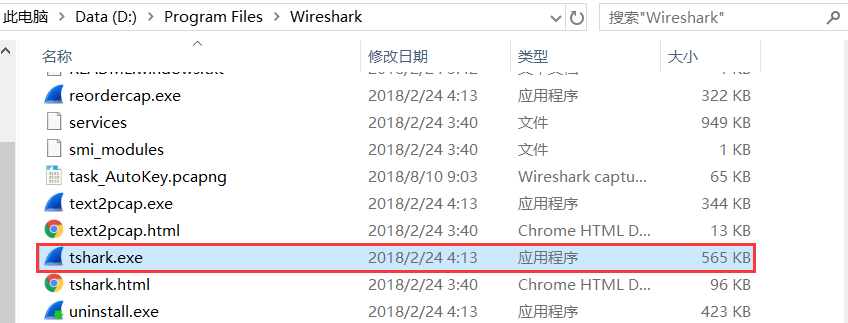
我们先拿 GitHub 上一个简单的例子开始讲起：



我们分析可以知道， USB 协议的数据部分在 Leftover Capture Data 域之中，在 Mac 和 Linux 下可以用 tshark 命令可以将 leftover capture data 单独提取出来，命令如下：

tshark -r example.pcap -T fields -e usb.capdata //如果想导入usbdata.txt文件中，后面加上参数：>usbdata.txt

Windows 下装了 wireshark 的环境下，在 wireshark目录下有个 tshark.exe ，比如我的在 D:\Program Files\Wireshark\tshark.exe

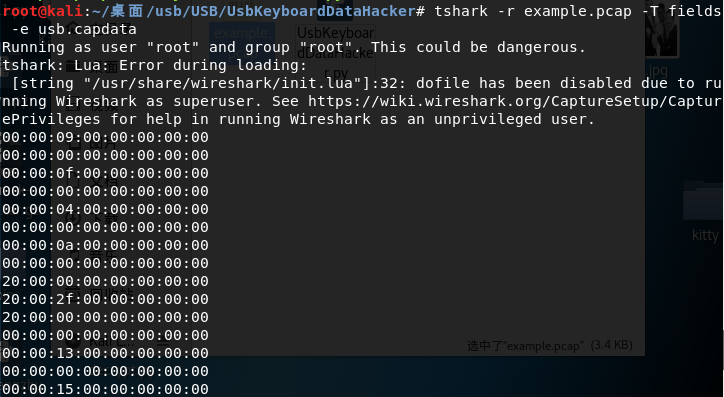


调用 cmd ，定位到当前目录下，输入如下命令即可：

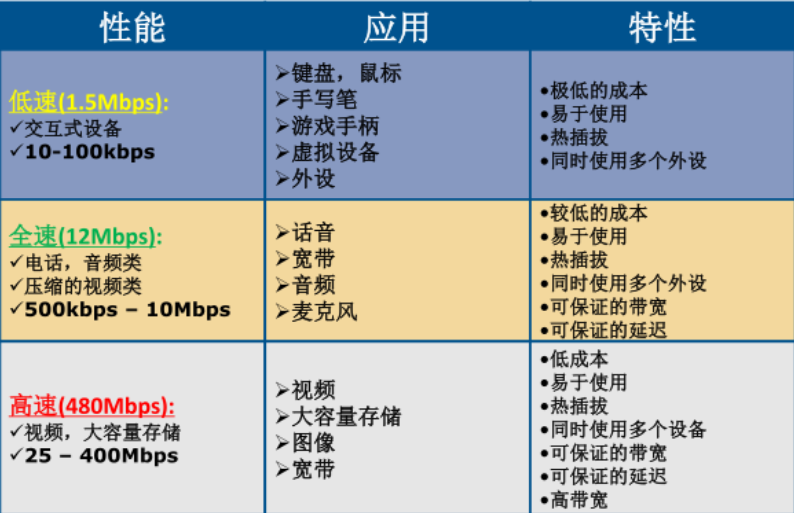
tshark.exe -r example.pcap -T fields -e usb.capdata //如果想导入usbdata.txt文件中，后面加上参数：>usbdata.txt

有关 tshark 命令的详细使用参考 wireshark 官方文档：<https://www.wireshark.org/docs/man-pages/tshark.html>

运行命令并查看 usbdata.txt 发现数据包长度为八个字节



关于 USB 的特点应用我找了一张图，很清楚的反应了这个问题：



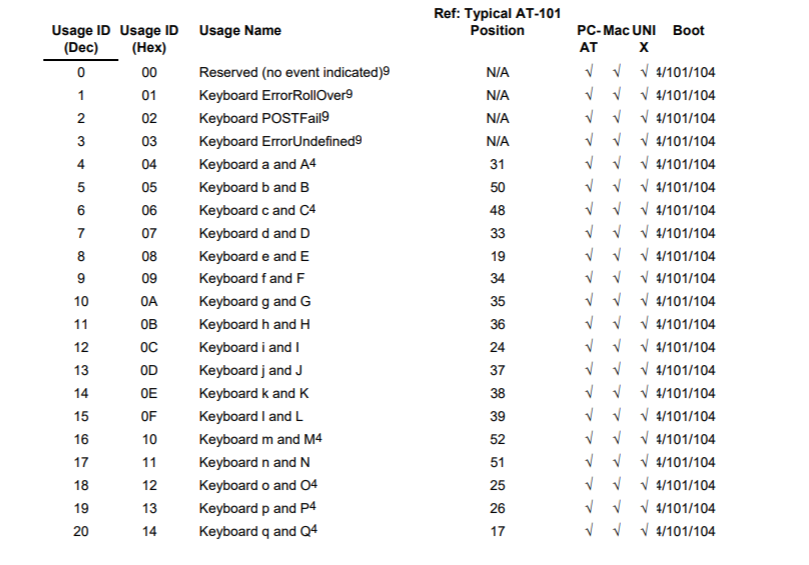
这里我们只关注 USB 流量中的键盘流量和鼠标流量。

键盘数据包的数据长度为 8 个字节，击键信息集中在第 3 个字节，每次 key stroke 都会产生一个 keyboard event usb packet 。

鼠标数据包的数据长度为 4 个字节，第一个字节代表按键，当取 0x00 时，代表没有按键、为 0x01 时，代表按左键，为 0x02 时，代表当前按键为右键。第二个字节可以看成是一个 signed byte 类型，其最高位为符号位，当这个值为正时，代表鼠标水平右移多少像素，为负时，代表水平左移多少像素。第三个字节与第二字节类似，代表垂直上下移动的偏移。

我翻阅了大量的 USB 协议的文档，在这里我们可以找到这个值与具体键位的对应关系：<https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf>

usb keyboard 的映射表 根据这个映射表将第三个字节取出来，对应对照表得到解码：



我们写出如下脚本：

mappings = { 0x04:"A", 0x05:"B", 0x06:"C", 0x07:"D", 0x08:"E", 0x09:"F", 0x0A:"G", 0x0B:"H", 0x0C:"I", 0x0D:"J", 0x0E:"K", 0x0F:"L", 0x10:"M", 0x11:"N",0x12:"O", 0x13:"P", 0x14:"Q", 0x15:"R", 0x16:"S", 0x17:"T", 0x18:"U",0x19:"V", 0x1A:"W", 0x1B:"X", 0x1C:"Y", 0x1D:"Z", 0x1E:"1", 0x1F:"2", 0x20:"3", 0x21:"4", 0x22:"5", 0x23:"6", 0x24:"7", 0x25:"8", 0x26:"9", 0x27:"0", 0x28:"n", 0x2a:"[DEL]", 0X2B:" ", 0x2C:" ", 0x2D:"-", 0x2E:"=", 0x2F:"[", 0x30:"]", 0x31:"\\", 0x32:"~", 0x33:";", 0x34:"'", 0x36:",", 0x37:"." }

nums = []

keys = open('usbdata.txt')

for line in keys:

if line[0]!='0' or line[1]!='0' or line[3]!='0' or line[4]!='0' or line[9]!='0' or line[10]!='0' or line[12]!='0' or line[13]!='0' or line[15]!='0' or line[16]!='0' or line[18]!='0' or line[19]!='0' or line[21]!='0' or line[22]!='0':

continue

nums.append(int(line[6:8],16))

# 00:00:xx:....

keys.close()

output = ""

for n in nums:

if n == 0 :

continue

if n in mappings:

output += mappings[n]

else:

output += '[unknown]'

print('output :n' + output)

结果如下：

IMG_256

我们把前面的整合成脚本，得：

#!/usr/bin/env python

import sys

import os

DataFileName = "usb.dat"

presses = []

normalKeys = {"04":"a", "05":"b", "06":"c", "07":"d", "08":"e", "09":"f", "0a":"g", "0b":"h", "0c":"i", "0d":"j", "0e":"k", "0f":"l", "10":"m", "11":"n", "12":"o", "13":"p", "14":"q", "15":"r", "16":"s", "17":"t", "18":"u", "19":"v", "1a":"w", "1b":"x", "1c":"y", "1d":"z","1e":"1", "1f":"2", "20":"3", "21":"4", "22":"5", "23":"6","24":"7","25":"8","26":"9","27":"0","28":"<RET>","29":"<ESC>","2a":"<DEL>", "2b":"\t","2c":"<SPACE>","2d":"-","2e":"=","2f":"[","30":"]","31":"\\","32":"<NON>","33":";","34":"'","35":"<GA>","36":",","37":".","38":"/","39":"<CAP>","3a":"<F1>","3b":"<F2>", "3c":"<F3>","3d":"<F4>","3e":"<F5>","3f":"<F6>","40":"<F7>","41":"<F8>","42":"<F9>","43":"<F10>","44":"<F11>","45":"<F12>"}

shiftKeys = {"04":"A", "05":"B", "06":"C", "07":"D", "08":"E", "09":"F", "0a":"G", "0b":"H", "0c":"I", "0d":"J", "0e":"K", "0f":"L", "10":"M", "11":"N", "12":"O", "13":"P", "14":"Q", "15":"R", "16":"S", "17":"T", "18":"U", "19":"V", "1a":"W", "1b":"X", "1c":"Y", "1d":"Z","1e":"!", "1f":"@", "20":"#", "21":"$", "22":"%", "23":"^","24":"&","25":"\*","26":"(","27":")","28":"<RET>","29":"<ESC>","2a":"<DEL>", "2b":"\t","2c":"<SPACE>","2d":"\_","2e":"+","2f":"{","30":"}","31":"|","32":"<NON>","33":"\"","34":":","35":"<GA>","36":"<","37":">","38":"?","39":"<CAP>","3a":"<F1>","3b":"<F2>", "3c":"<F3>","3d":"<F4>","3e":"<F5>","3f":"<F6>","40":"<F7>","41":"<F8>","42":"<F9>","43":"<F10>","44":"<F11>","45":"<F12>"}

def main():

# check argv

if len(sys.argv) != 2:

print "Usage : "

print " python UsbKeyboardHacker.py data.pcap"

print "Tips : "

print " To use this python script , you must install the tshark first."

print " You can use `sudo apt-get install tshark` to install it"

print " Thank you for using."

exit(1)

# get argv

pcapFilePath = sys.argv[1]

# get data of pcap

os.system("tshark -r %s -T fields -e usb.capdata > %s" % (pcapFilePath, DataFileName))

# read data

with open(DataFileName, "r") as f:

for line in f:

presses.append(line[0:-1])

# handle

result = ""

for press in presses:

Bytes = press.split(":")

if Bytes[0] == "00":

if Bytes[2] != "00":

result += normalKeys[Bytes[2]]

elif Bytes[0] == "20": # shift key is pressed.

if Bytes[2] != "00":

result += shiftKeys[Bytes[2]]

else:

print "[-] Unknow Key : %s" % (Bytes[0])

print "[+] Found : %s" % (result)

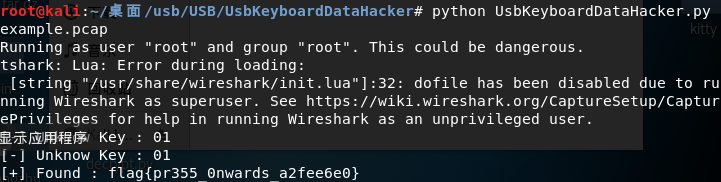
# clean the temp data

os.system("rm ./%s" % (DataFileName))

if \_\_name\_\_ == "\_\_main\_\_":

main()

效果如下：



另外贴上一份鼠标流量数据包转换脚本：

nums = []

keys = open('usbdata.txt','r')

posx = 0

posy = 0

for line in keys:

if len(line) != 12 :

continue

x = int(line[3:5],16)

y = int(line[6:8],16)

if x > 127 :

x -= 256

if y > 127 :

y -= 256

posx += x

posy += y

btn\_flag = int(line[0:2],16) # 1 for left , 2 for right , 0 for nothing

if btn\_flag == 1 :

print posx , posy

keys.close()

键盘流量数据包转换脚本如下：

nums=[0x66,0x30,0x39,0x65,0x35,0x34,0x63,0x31,0x62,0x61,0x64,0x32,0x78,0x33,0x38,0x6d,0x76,0x79,0x67,0x37,0x77,0x7a,0x6c,0x73,0x75,0x68,0x6b,0x69,0x6a,0x6e,0x6f,0x70]

s=''

for x in nums:

s+=chr(x)

print s

mappings = { 0x41:"A", 0x42:"B", 0x43:"C", 0x44:"D", 0x45:"E", 0x46:"F", 0x47:"G", 0x48:"H", 0x49:"I", 0x4a:"J", 0x4b:"K", 0x4c:"L", 0x4d:"M", 0x4e:"N",0x4f:"O", 0x50:"P", 0x51:"Q", 0x52:"R", 0x53:"S", 0x54:"T", 0x55:"U",0x56:"V", 0x57:"W", 0x58:"X", 0x59:"Y", 0x5a:"Z", 0x60:"0", 0x61:"1", 0x62:"2", 0x63:"3", 0x64:"4", 0x65:"5", 0x66:"6", 0x67:"7", 0x68:"8", 0x69:"9", 0x6a:"\*", 0x6b:"+", 0X6c:"separator", 0x6d:"-", 0x6e:".", 0x6f:"/" }

output = ""

for n in nums:

if n == 0 :

continue

if n in mappings:

output += mappings[n]

else:

output += '[unknown]'

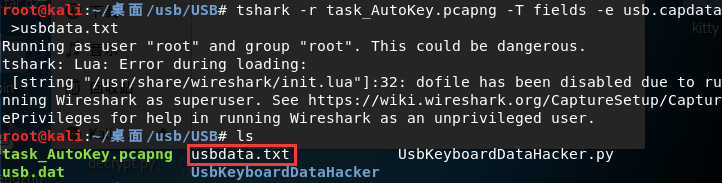
print 'output :\n' + output

那么对于 xman 三期夏令营排位赛的这道题，我们可以模仿尝试如上这个例子：

首先我们通过 tshark 将 usb.capdata 全部导出：

tshark -r task\_AutoKey.pcapng -T fields -e usb.capdata //如果想导入usbdata.txt文件中，后面加上参数：>usbdata.txt

结果如下：



我们用上面的 python 脚本将第三个字节取出来，对应对照表得到解码：

mappings = { 0x04:"A", 0x05:"B", 0x06:"C", 0x07:"D", 0x08:"E", 0x09:"F", 0x0A:"G", 0x0B:"H", 0x0C:"I", 0x0D:"J", 0x0E:"K", 0x0F:"L", 0x10:"M", 0x11:"N",0x12:"O", 0x13:"P", 0x14:"Q", 0x15:"R", 0x16:"S", 0x17:"T", 0x18:"U",0x19:"V", 0x1A:"W", 0x1B:"X", 0x1C:"Y", 0x1D:"Z", 0x1E:"1", 0x1F:"2", 0x20:"3", 0x21:"4", 0x22:"5", 0x23:"6", 0x24:"7", 0x25:"8", 0x26:"9", 0x27:"0", 0x28:"n", 0x2a:"[DEL]", 0X2B:" ", 0x2C:" ", 0x2D:"-", 0x2E:"=", 0x2F:"[", 0x30:"]", 0x31:"\\", 0x32:"~", 0x33:";", 0x34:"'", 0x36:",", 0x37:"." }

nums = []

keys = open('usbdata.txt')

for line in keys:

if line[0]!='0' or line[1]!='0' or line[3]!='0' or line[4]!='0' or line[9]!='0' or line[10]!='0' or line[12]!='0' or line[13]!='0' or line[15]!='0' or line[16]!='0' or line[18]!='0' or line[19]!='0' or line[21]!='0' or line[22]!='0':

continue

nums.append(int(line[6:8],16))

# 00:00:xx:....

keys.close()

output = ""

for n in nums:

if n == 0 :

continue

if n in mappings:

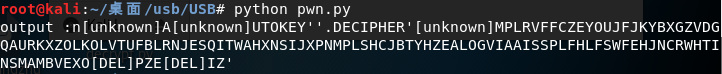
output += mappings[n]

else:

output += '[unknown]'

print('output :n' + output)

运行结果如下：



output :n[unknown]A[unknown]UTOKEY''.DECIPHER'[unknown]MPLRVFFCZEYOUJFJKYBXGZVDGQAURKXZOLKOLVTUFBLRNJESQITWAHXNSIJXPNMPLSHCJBTYHZEALOGVIAAISSPLFHLFSWFEHJNCRWHTINSMAMBVEXO[DEL]PZE[DEL]IZ'

我们可以看出这是自动密匙解码，现在的问题是在我们不知道密钥的情况下应该如何解码呢？

我找到了如下这篇关于如何爆破密匙：<http://www.practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-autokey-cipher/>

爆破脚本如下：

from ngram\_score import ngram\_score

from pycipher import Autokey

import re

from itertools import permutations

qgram = ngram\_score('quadgrams.txt')

trigram = ngram\_score('trigrams.txt')

ctext = 'MPLRVFFCZEYOUJFJKYBXGZVDGQAURKXZOLKOLVTUFBLRNJESQITWAHXNSIJXPNMPLSHCJBTYHZEALOGVIAAISSPLFHLFSWFEHJNCRWHTINSMAMBVEXPZIZ'

ctext = re.sub(r'[^A-Z]','',ctext.upper())

# keep a list of the N best things we have seen, discard anything else

class nbest(object):

def \_\_init\_\_(self,N=1000):

self.store = []

self.N = N

def add(self,item):

self.store.append(item)

self.store.sort(reverse=True)

self.store = self.store[:self.N]

def \_\_getitem\_\_(self,k):

return self.store[k]

def \_\_len\_\_(self):

return len(self.store)

#init

N=100

for KLEN in range(3,20):

rec = nbest(N)

for i in permutations('ABCDEFGHIJKLMNOPQRSTUVWXYZ',3):

key = ''.join(i) + 'A'\*(KLEN-len(i))

pt = Autokey(key).decipher(ctext)

score = 0

for j in range(0,len(ctext),KLEN):

score += trigram.score(pt[j:j+3])

rec.add((score,''.join(i),pt[:30]))

next\_rec = nbest(N)

for i in range(0,KLEN-3):

for k in xrange(N):

for c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':

key = rec[k][1] + c

fullkey = key + 'A'\*(KLEN-len(key))

pt = Autokey(fullkey).decipher(ctext)

score = 0

for j in range(0,len(ctext),KLEN):

score += qgram.score(pt[j:j+len(key)])

next\_rec.add((score,key,pt[:30]))

rec = next\_rec

next\_rec = nbest(N)

bestkey = rec[0][1]

pt = Autokey(bestkey).decipher(ctext)

bestscore = qgram.score(pt)

for i in range(N):

pt = Autokey(rec[i][1]).decipher(ctext)

score = qgram.score(pt)

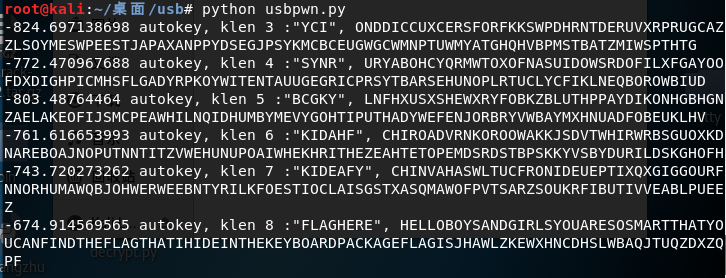
if score > bestscore:

bestkey = rec[i][1]

bestscore = score

print bestscore,'autokey, klen',KLEN,':"'+bestkey+'",',Autokey(bestkey).decipher(ctext)

跑出来的结果如下：



我们看到了 flag 的字样，整理可得如下：

-674.914569565 autokey, klen 8 :"FLAGHERE", HELLOBOYSANDGIRLSYOUARESOSMARTTHATYOUCANFINDTHEFLAGTHATIHIDEINTHEKEYBOARDPACKAGEFLAGISJHAWLZKEWXHNCDHSLWBAQJTUQZDXZQPF

我们把字段进行分割看：

HELLO BOYS AND GIRLS YOU ARE SO SMART THAT YOU CAN FIND THE FLAG THAT IH IDE IN THE KEY BOARD PACKAGE FLAG IS JHAWLZKEWXHNCDHSLWBAQJTUQZDXZQPF

最后的 flag 就是 flag{JHAWLZKEWXHNCDHSLWBAQJTUQZDXZQPF}

## 参考文献

* <https://www.cnblogs.com/ECJTUACM-873284962/p/9473808.html>
* <https://blog.csdn.net/songze_lee/article/details/77658094>
* <https://wiki.wireshark.org/USB>
* <https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf>
* <https://www.wireshark.org/docs/man-pages/tshark.html>
* <http://www.practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-autokey-cipher/>
* <https://hackfun.org/2017/02/22/CTF%E4%B8%AD%E9%82%A3%E4%BA%9B%E8%84%91%E6%B4%9E%E5%A4%A7%E5%BC%80%E7%9A%84%E7%BC%96%E7%A0%81%E5%92%8C%E5%8A%A0%E5%AF%86/>

1. **数据提取**

这一块是流量包中另一个重点, 通过对协议分析, 找到了题目的关键点, 如何提取数据成了接下来的关键问题

## wireshark

### wireshark 自动分析

file -> export objects -> http

### 手动数据提取

file->export selected Packet Bytes

## tshark

tshark 作为 wireshark 的命令行版, 高效快捷是它的优点, 配合其余命令行工具 (awk,grep) 等灵活使用, 可以快速定位, 提取数据从而省去了繁杂的脚本编写

再看Google CTF 2016 Forensic-200这一题, 可以通过 tshark 迅速完成解题

what@kali:/tmp$ tshark -r capture.pcapng -T fields -e usb.capdata > data2.txt

what@kali:/tmp$ # awk -F: 'function comp(v){if(v>127)v-=256;return v}{x+=comp(strtonum("0x"$2));y+=comp(strtonum("0x"$3))}$1=="01"{print x,y}' data.txt > data3.txt

what@kali:/tmp$ gnuplot

> plot "data3.txt"

* Step 1 鼠标协议中数据提取
* Step 2 通过 awk 进行位置坐标转换
* Step 3 形成图形

### 常用方法

tshark -r \*\*.pcap –Y \*\* -T fields –e \*\* | \*\*\*\* > data

Usage:

-Y <display filter> packet displaY filter in Wireshark display filter

syntax

-T pdml|ps|psml|json|jsonraw|ek|tabs|text|fields|?

format of text output (def: text)

-e <field> field to print if -Tfields selected (e.g. tcp.port,

\_ws.col.Info)

通过-Y过滤器 (与 wireshark 一致), 然后用-T filds -e配合指定显示的数据段 (比如 usb.capdata)

* tips
  + -e后的参数不确定可以由 wireshark 右击需要的数据选中后得到

### 例题

题目：google-ctf-2016 : a-cute-stegosaurus-100

这题的数据隐藏的非常巧妙, 而且有一张图片混淆视听, 需要对tcp协议非常熟悉, 所以当时做出来的人并不多, 全球只有 26 支队伍

在tcp报文段中有 6Bit 的状态控制码, 分别如下

* URG：紧急比特（urgent）, 当 URG＝1 时，表明紧急指针字段有效, 代表该封包为紧急封包。它告诉系统此报文段中有紧急数据，应尽快传送 (相当于高优先级的数据)
* ACK：确认比特（Acknowledge）。只有当 ACK＝1 时确认号字段才有效, 代表这个封包为确认封包。当 ACK＝0 时，确认号无效。
* PSH：（Push function）若为 1 时，代表要求对方立即传送缓冲区内的其他对应封包，而无需等缓冲满了才送。
* RST：复位比特 (Reset) , 当 RST＝1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。
* SYN：同步比特 (Synchronous)，SYN 置为 1，就表示这是一个连接请求或连接接受报文, 通常带有 SYN 标志的封包表示『主动』要连接到对方的意思。。
* FIN：终止比特 (Final)，用来释放一个连接。当 FIN＝1 时，表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。

而这里的tcp.urg却为



通过 tshark 提取tcp.urg然后去除 0 的字段, 换行符转,直接转换成 python 的列表, 转 ascii 即可得到 flag

⚡ root@kali: tshark -r Stego-200\_urg.pcap -T fields -e tcp.urgent\_pointer|egrep -vi "^0$"|tr '\n' ','

Running as user "root" and group "root". This could be dangerous.

67,84,70,123,65,110,100,95,89,111,117,95,84,104,111,117,103,104,116,95,73,116,95,87,97,115,95,73,110,95,84,104,101,95,80,105,99,116,117,114,101,125,#

...

>>> print "".join([chr(x) for x in arr]) #python转换ascii

CTF{And\_You\_Thought\_It\_Was\_In\_The\_Picture}

题目：stego-150\_ears.xz

**Step 1**

通过file命令不断解压得到 pcap 文件

➜ Desktop file ears

ears: XZ compressed data

➜ Desktop unxz < ears > file\_1

➜ Desktop file file\_1

file\_1: POSIX tar archive

➜ Desktop 7z x file\_1

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21

p7zip Version 16.02 (locale=en\_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,1 CPU Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz (306C3),ASM,AES-NI)

Scanning the drive for archives:

1 file, 4263936 bytes (4164 KiB)

Extracting archive: file\_1

--

Path = file\_1

Type = tar

Physical Size = 4263936

Headers Size = 1536

Code Page = UTF-8

Everything is Ok

Size: 4262272

Compressed: 4263936

**Step 2**

通过 wireshark 发现 dns 中回应名字存在异常，组成 16 进制的 png 文件

采用 tshark 进行提取，提取 dns 中的数据, 筛选具体报文形式\w{4,}.asis.io

tshark -r forensic\_175\_d78a42edc01c9104653776f16813d9e5 -T fields -e dns.qry.name -e dns.flags|grep 8180|awk '{if ($1~/\w{4,}.asis.io/) print $1}'|awk -F '.' '{print $1}'|tr -d '\n' > png

**Step 3**

16 进制还原图片

xxd -p -r png flag

## 自定义协议

提取数据存在一类特殊情况，即传输的数据本身使用自定义协议，下面用 HITCON 2018 的两道 Misc 为例说明。

### 例题分析

[HITCON-2018 : ev3 basic](https://github.com/ctf-wiki/ctf-challenges/tree/master/misc/cap/2018HITCON-ev3-basic)

[HITCON-2018 : ev3 scanner](https://github.com/ctf-wiki/ctf-challenges/tree/master/misc/cap/2018HITCON-ev3-scanner)

**ev3 basic**

#### **确定数据**

对于这类题目，首先分析有效数据位于哪些包中。观察流量，通讯双方为 localhost 和 LegoSystem 。其中大量标为 PKTLOG 的数据包都是日志，此题中不需关注。简单浏览其余各个协议的流量，发现仅 RFCOMM 协议中存在没有被 wireshark 解析的 data 段，而 RFCOMM 正是蓝牙使用的[传输层协议](https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols" \l "Radio_frequency_communication_(RFCOMM))之一。

由前述 tshark 相关介绍，可以通过以下命令提取数据：

tshark -r .\ev3\_basic.pklg -T fields -e data -Y "btrfcomm"

#### **分析协议**

找到数据后，需要确定数据格式。如何查找资料可以参考 信息搜集技术 一节，此处不再赘述。总之由 ev3 这个关键词出发，我们最终知道这种通信方式传输的内容被称之为 [Direct Command](http://ev3directcommands.blogspot.com/2016/01/no-title-specified-page-table-border_94.html)，所使用的是乐高自定义的一种[简单应用层协议](https://le-www-live-s.legocdn.com/sc/media/files/ev3-developer-kit/lego mindstorms ev3 communication developer kit-f691e7ad1e0c28a4cfb0835993d76ae3.pdf?la=en-us)，Command 本身格式由乐高的手册 [EV3 Firmware Developer Kit](http://www.lego.com/en-gb/mindstorms/downloads) 定义。（查找过程并不像此处简单而直观，也是本题的关键点之一。）

在乐高的协议中，发送和回复遵从不同格式。在 ev3 basic 中，所有回复流量都相同，通过手册可知内容代表 ok ，没有实际含义，而发送的每个数据包都包含了一条指令。由协议格式解析出指令的 Opcode 均为 0x84 ，代表 UI\_DRAW 函数，且 CMD 是 0x05 ，代表 TEXT 。之后是四个参数，Color, X0, Y0, STRING 。此处需要注意乐高的单个参数字节数并不固定，即便手册上标明了数据类型是 DATA16 ，仍然可能使用一个字节长度的参数，需要参照手册中 Parameter encoding 一节及[相关文章](http://ev3directcommands.blogspot.com/2016/01/ev3-direct-commands-lesson-02-pre.html)。

尝试分析几个命令，发现每个指令都会在屏幕特定位置打印一个字符，这与提供的图片相符。

#### **处理结果**

理解数据内容后，通过脚本提取所有命令并解析参数，需要注意单个参数的字节数不固定。

得到所有命令的参数后，可以将每个字符其按照坐标绘制在屏幕上。较简单的做法是先按 X 后按 Y 排序，直接输出即可。

**ev3 scanner**

第二题的做法与第一题基本相同，难度增加的地方在于：

1.发送的命令不再单一，包括读取传感器信息、控制 ev3 运动

2.回复也包含信息，主要是传感器读取的内容

3.函数的参数更复杂，解析难度更大

4.解析命令得到的结果需要更多处理

ev3 scanner 此处不再提供详细方法，可作为练习加深对这一类型题目的理解。

### Python Script

TODO