CTF——MISC——流量分析

原创

Captain Hammer 于 2019-09-26 14:43:23 发布 15684 收藏 91
分类专栏： CTF 类型题总结

本文链接：https://blog.csdn.net/vhkjhwbs/article/details/100847221

目录

概括来讲在比赛中的流量分析有以下三个方向：
1、流量包修复
2、协议分析
3、数据提取

一、流量包修复

比如一个流量包它的文件头也是对的，里边也没有包含其他的文件等等等等，但是就是打开出现一些未知的错误，这时候就要考虑对流量包进行修复。

这类题目考察较少，通常都借助现成的工具例如PCAPFIX直接修复。

PcapFix

二、协议分析

此类方向需要对分析流量包工具所用的语法有一定的掌握，这里以wireshark为例，须掌握wireshark过滤器（捕捉过滤器与显示过滤器）的基础语法，从而更快更精准的获取指定的信息。

捕捉过滤器：用于决定将什么样的信息记录在捕捉结果中，需要在开始捕捉前设置。
显示过滤器：用于在捕获结果中进行详细查找，可以在得到捕捉结果后进行更改

捕捉过滤器基础语法

Protocol Direction Host(s) Value LogicalOperations other expression

tcp dst 10.1.1.1 80 and tcp dst 10.2.2.2 3128

Protocol

可能的值: ether, fddi, ip, arp, rarp, decnet,lat, sca, moprc, mopdl, tcp and udp，如果没有特别指明是什么协议，则默认使用所有支持的协议。

Direction

可能的值: src, dst, src and dst, src or dst，如果没有特别指明来源或目的地，则默认使用 “src or dst” 作为关键字。

Host(s)

可能的值： net, port,host, portrange，如果没有指定此值，则默认使用”host”关键字。

例如，”src 10.1.1.1”与”src host 10.1.1.1”相同。

Logical Operations

可能的值：not, and, or

否(“not”)具有最高的优先级，或(“or”)和与(“and”)具有相同的优先级

“not tcp port 3128 and tcp port23”与”(not tcp port 3128) and tcp port23”相同。

举例分析：

tcp dst port 3128 //目的TCP端口为3128的封包。

ip src host 10.1.1.1 //来源IP地址为10.1.1.1的封包。

host 10.1.2.3 //目的或来源IP地址为10.1.2.3的封包。

src portrange 2000-2500

//来源为UDP或TCP，并且端口号在2000至2500范围内的封包

not icmp //除了icmp以外的所有封包。

显示过滤器基础语法

Protocol String1 String2 ComparisonOperator Value Logical Operations other expression

Protocol

可以使用大量位于OSI模型第2至7层的协议。在Expression中可以看到，例如，IP，TCP，DNS，SSH

String1，String2

可选择显示过滤器右侧表达式，点击父类的+号，然后查看其子类

Comparison Oerators

可以使用六种比较运算符

英文写法：	C语言写法：	含义：
eq	==	等于
ne	!=	不等于
gt	>	大于
lt	<	小于
ge	>=	大于等于
le	<=	小于等于

英文写法：	C语言写法：	含义：
and	&&	逻辑与
or	||	逻辑或
xor	^^	逻辑异或
not	!	逻辑非

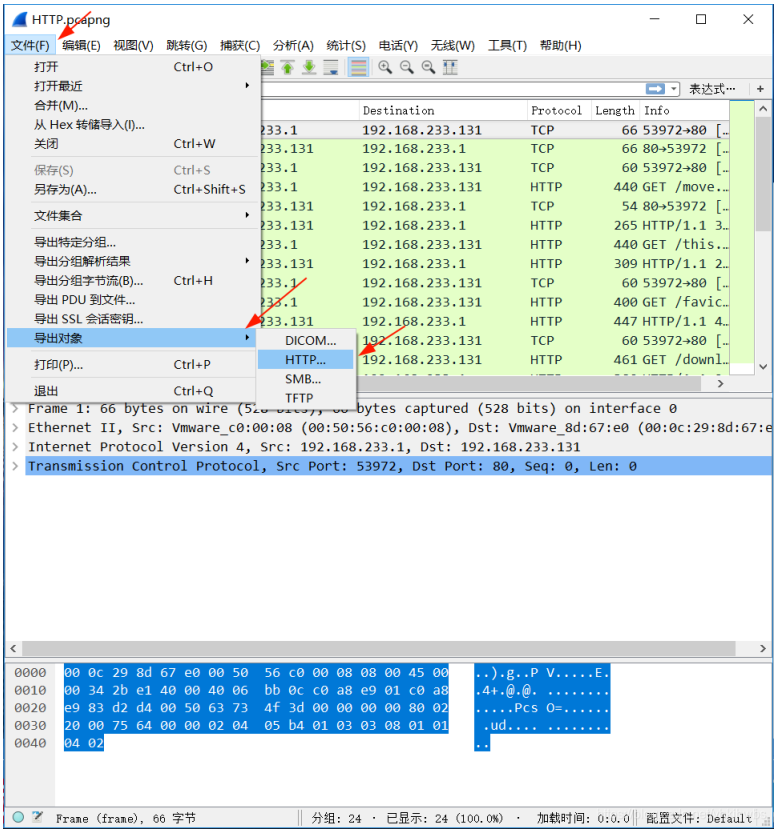
举例分析：
snmp || dns || icmp //显示SNMP或DNS或ICMP封包
ip.addr == 10.1.1.1 //显示源或目的IP为10.1.1.1的封包
ip.src != 10.1.2.3 and ip.dst!=10.4.5.6 //显示源不为10.1.2.3并且目的不为10.4.5.6的封包
tcp.port == 25 //显示来源或目的TCP端口号为25的封包
tcp.dport == 25 //显示目的TCP端口号为25的封包

如果过滤器语法是正确的，表达式背景为绿色，否则为红色

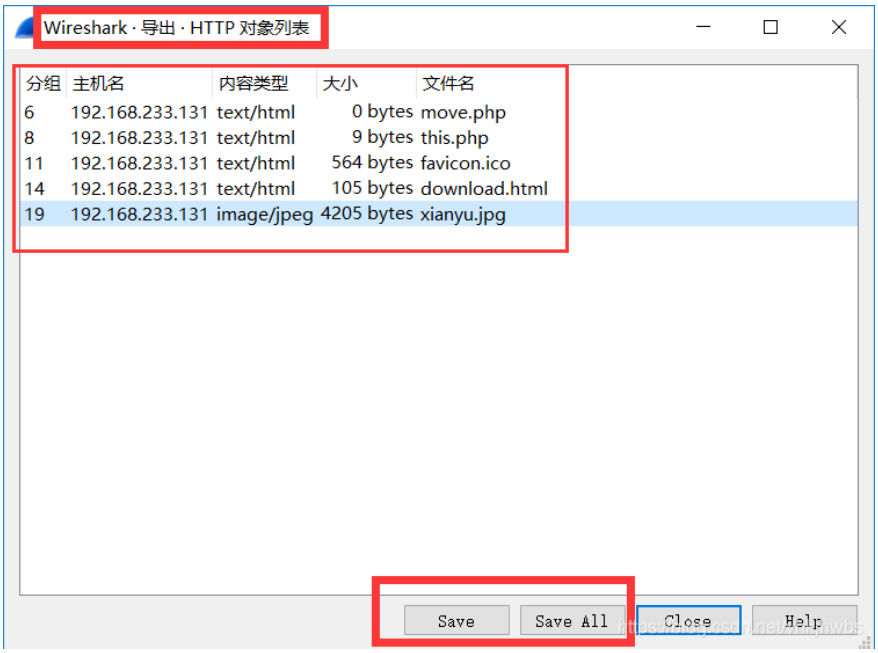
三、数据提取

这一块是流量包中另一个重点，通过对协议分析，找到题目的关键点，从而对所需要的数据进行提取。

Wireshark支持提取通过http传输（上传/下载）的文件内容，方法如下：
自动提取通过http传输的文件内容
文件->导出对象->HTTP



在打开的对象列表中找到有价值的文件，如压缩文件、文本文件、音频文件、图片等，点击Save进行保存，或者Save All保存所有对象再进入文件夹进行分析。



手动提取通过http传输的文件内容

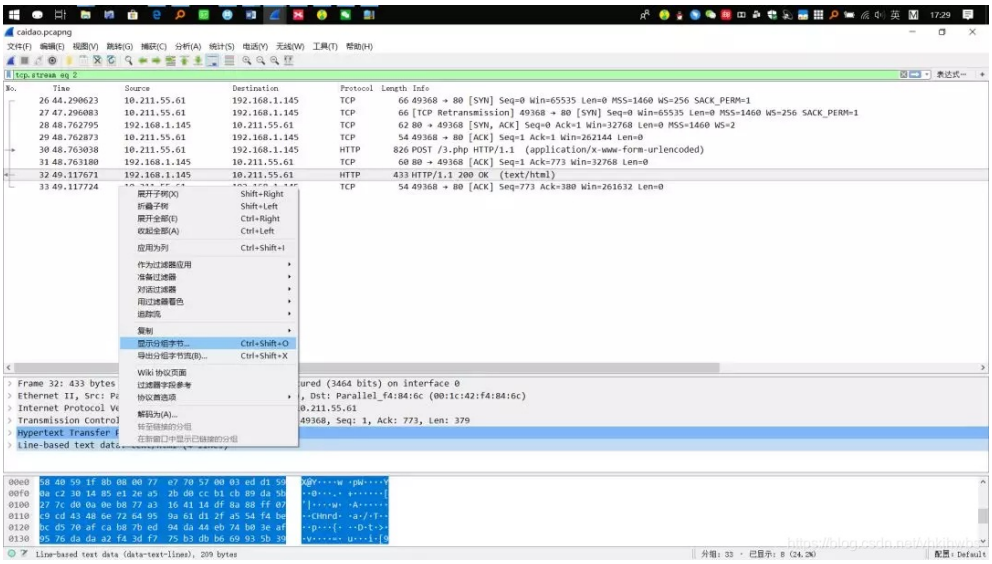
选中http文件传输流量包，在分组详情中找到`data,Line-based text, JPEG File Interchange Format, data:text/html`层，鼠标右键点击 – 选中 导出分组字节流。

来源: 简简

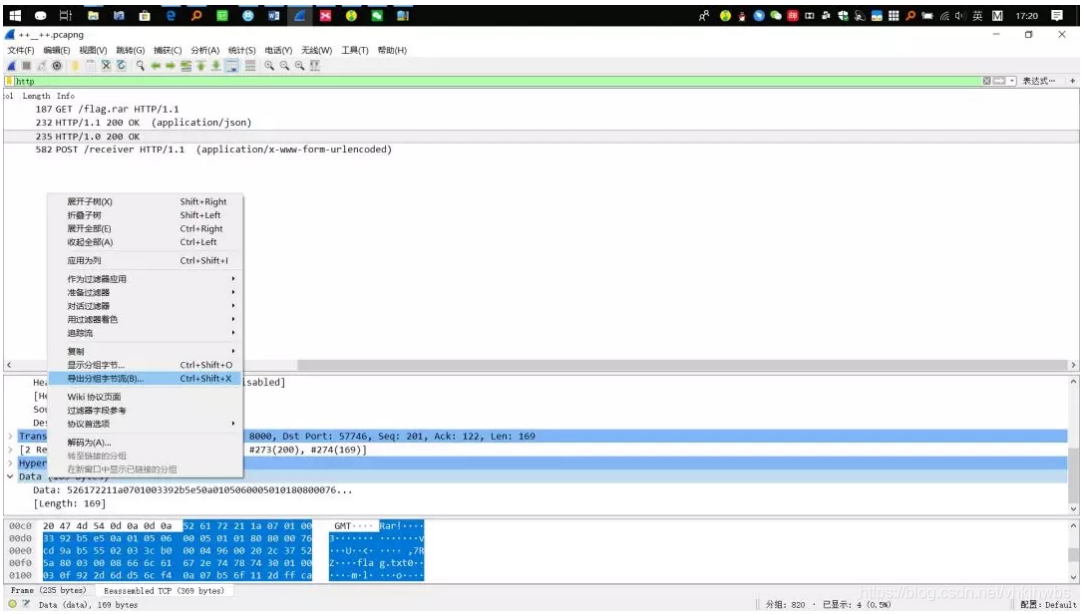
作者: 简简

链接: https://jwt1399.top/2019/07/29/ctf-liu-liang-fen-xi-zong-jie/

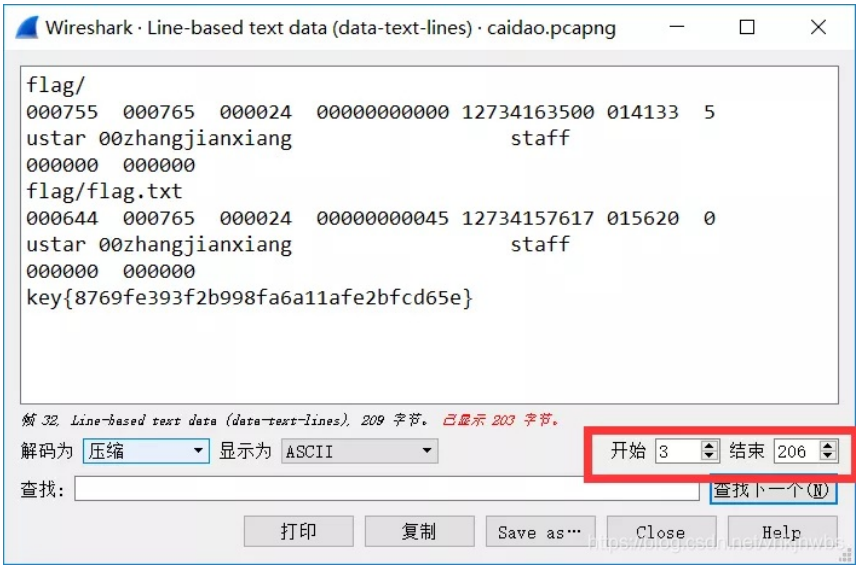
本文章著作权归作者所有，任何形式的转载都请注明出处。



如果是菜刀下载文件的流量，需要删除分组字节流前开头和结尾的X@Y字符，否则下载的文件会出错。鼠标右键点击 – 选中 显示分组字节



在弹出的窗口中设置开始和结束的字节（原字节数开头加3，结尾减3）



最后点击Save as按钮导出。

例题：

1，题目：Cephalopod(图片提取)

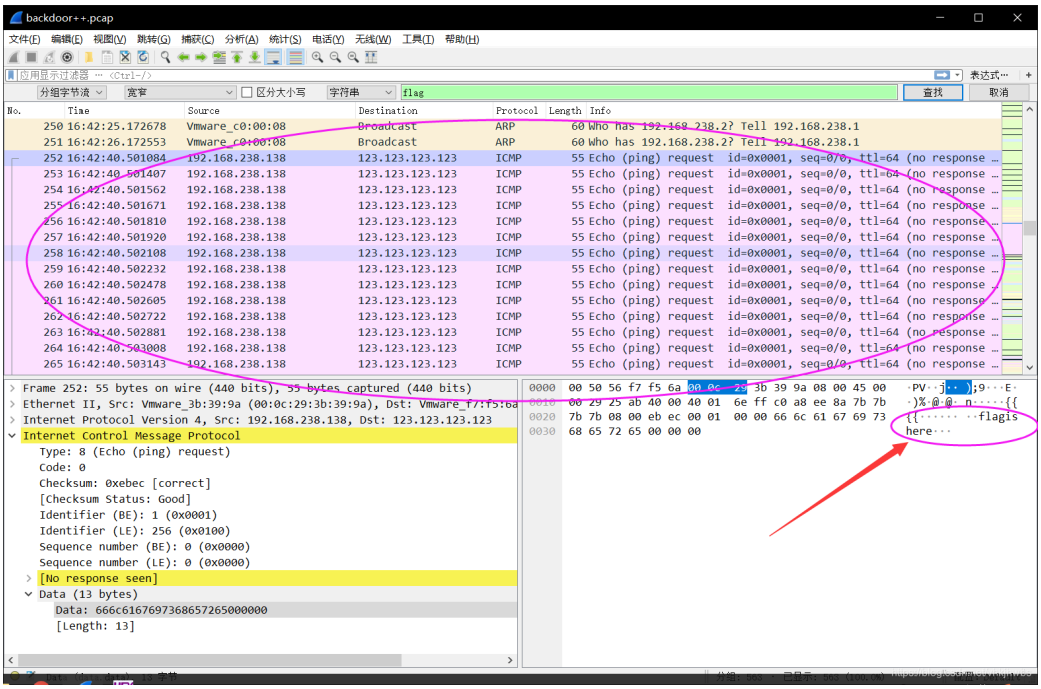
题目来源：XCTF 3rd-HITB CTF-2017
考点：图片提取
题目信息：(Cephalopod.pcapng)

c中zxz中z

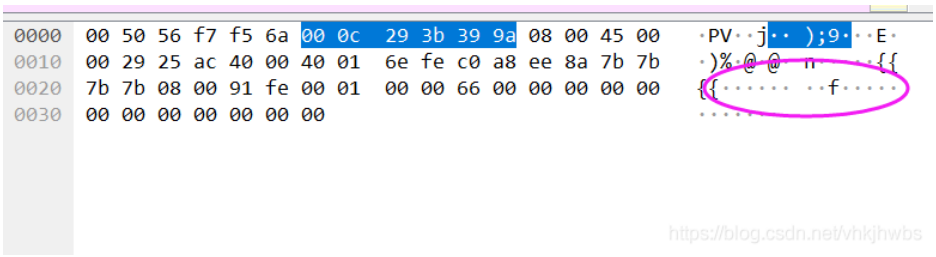
2,题目：特殊后门(icmp协议信息传输)

题目来源：第七届山东省大学生网络安全技能大赛
考点：字符串搜索，icmp协议信息传输
题目信息：(backdoor++.pcapng)

打卡数据包，先在字节流中 搜索 flag 字符串 ：搜索到了 一段连续的数据包 里面都有flag字符串



发现下面每一个 包里 都有一个 字符：



一个一个收集后得到：

flag{Icmp_backdoor_can_transfer-some_infomation}

小知识点：

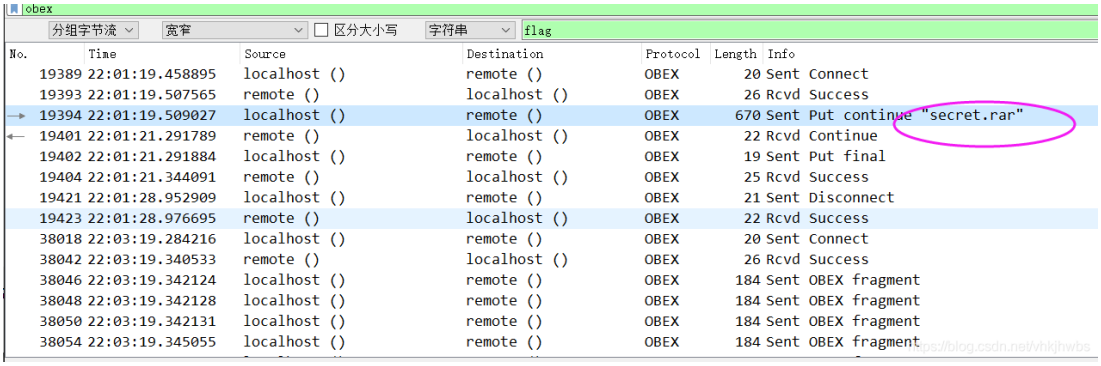
ICMP（Internet Control Message Protocol）Internet控制报文协议，它是TCP/IP协议簇的一个子协议，用于在IP主机、路由器之间传递控制消息。控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用，ICMP协议是一种面向无连接的协议，用于传输出错报告控制信息。它是一个非常重要的协议，它对于网络安全具有极其重要的意义。

3,题目：手机热点(蓝牙传输协议obex,数据提取)

题目来源：第七季极客大挑战
考点：蓝牙传输协议obex,数据提取
题目信息：(Blatand_1.pcapng)

题中说用 没流浪 向电脑传了文件 那肯定是 用的蓝牙 蓝牙协议 为 obex协议

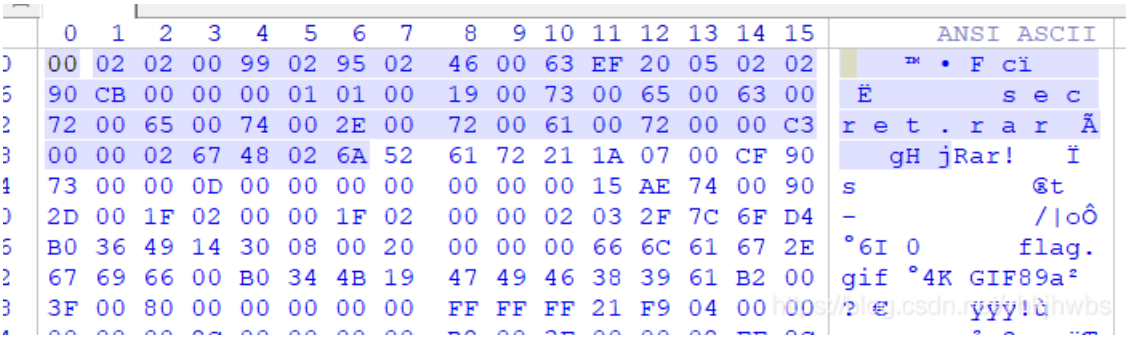
帅选出 obex 协议包 ：发现 有一个包 里有一个 rar压缩包：



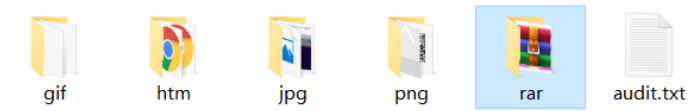
要分离出 这个 rar 包：

方法一：直接复制 数据块 然后复制到winhex中 转存为 rar

复制数据块 as Hex stream > 在winhex中粘贴 为 ASCII HEX > 删除前面的 多余信息 > 保存为 rar



方法二 ：用formost 分离：分离出来好多东西，不过我们目标是rar包

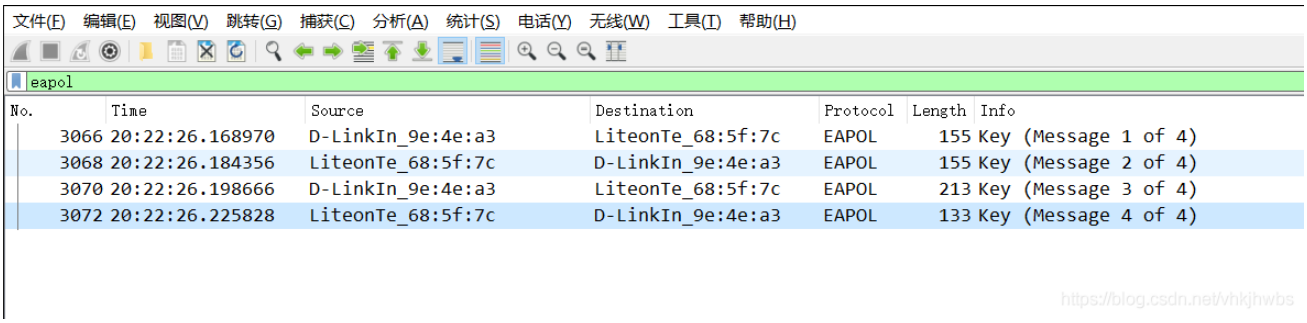


得到一张flag.git:

4,题目：想蹭网先解开密码(无线密码破解)

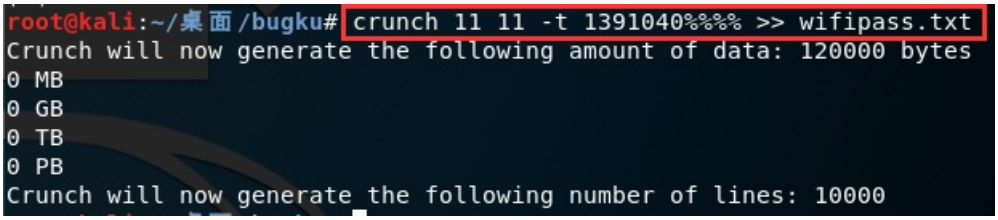
题目来源：bugku
考点：无线密码破解
题目信息：(wifi.cap)

wifi 连接认证 的重点在于 WAP的 四次握手过程 ， 就是 EAPOL 协议的包，

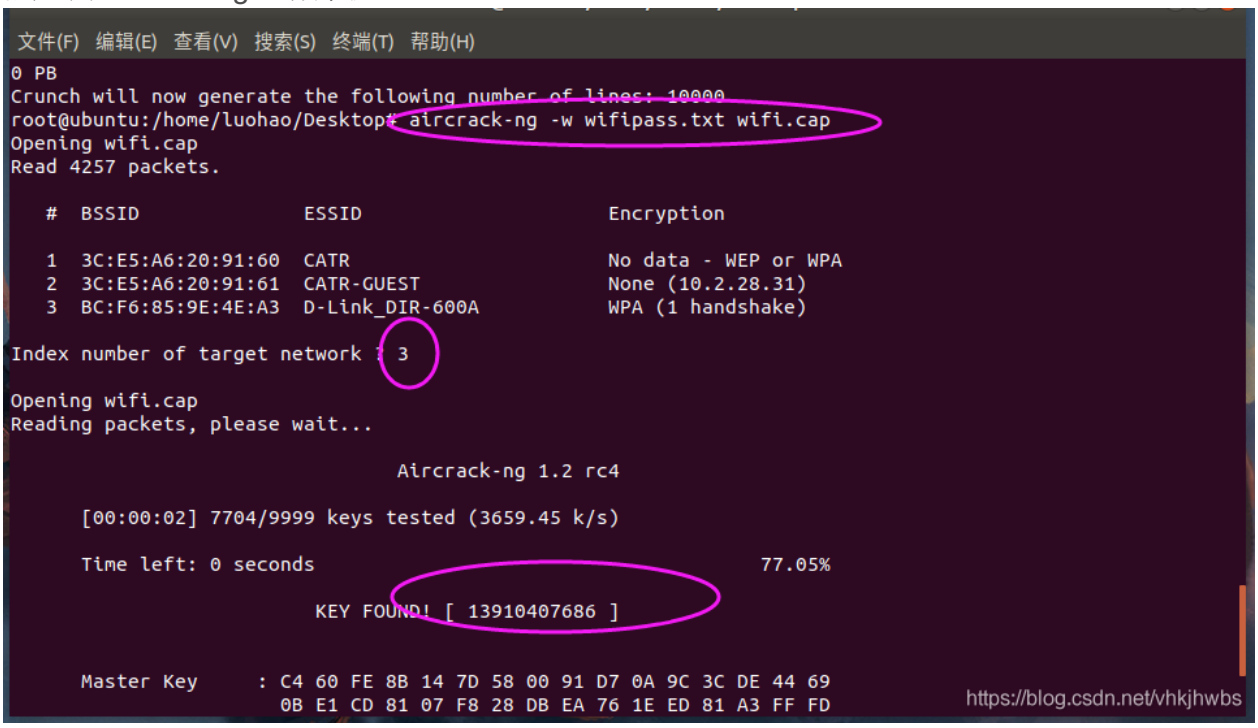


存在 握手过程：直接进行爆破：

先用 linux中的 crunch 生成一个字典：



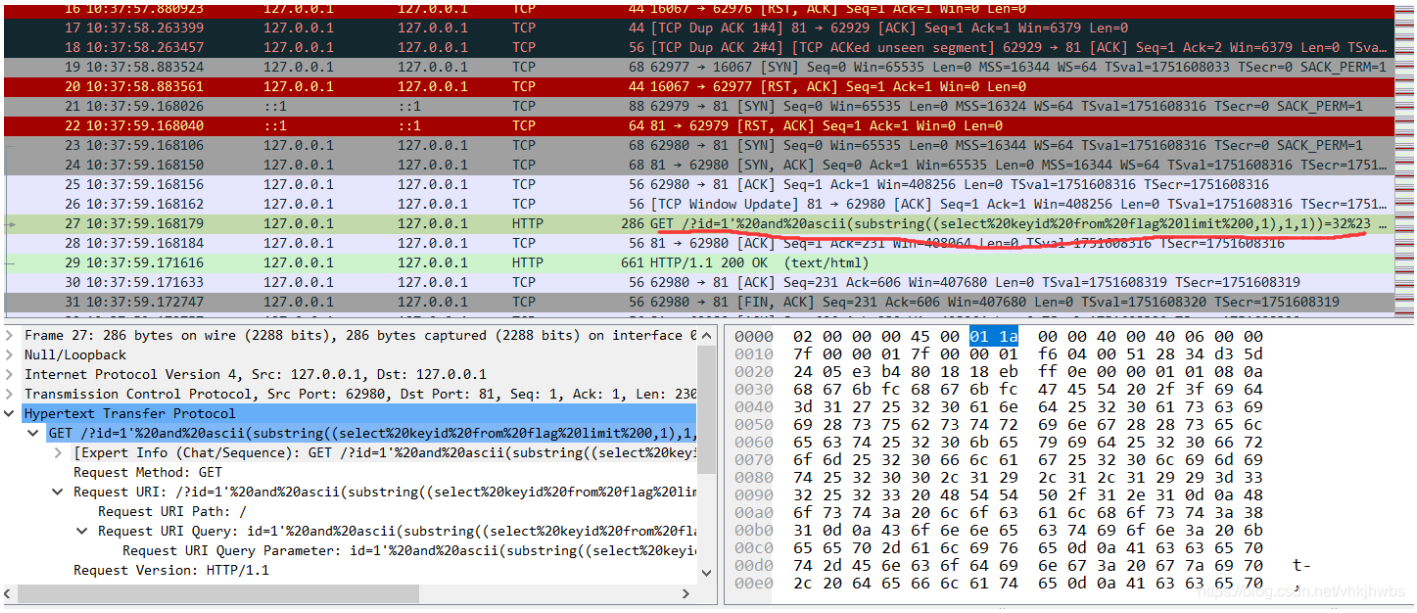
然后 用 aircrack-ng 进行爆破：



得到 WiFi密码

5,我的的教练也想打CTF

打开流量包，直接搜 字符串 flag ，找到很多 包，仔细看这些包的info ，发现是 sql 注入语句 布尔注入：



鉴于 sql 注入的 布尔注入 的原理：逐个字母爆破要查询的字符，我们可以把 查询到的每个字符 收集起来就能得到 flag了

注： 爆破每个字符的最后一个 查询语句 对应的字符就是 正确的字符

本题的爆破语句 ：

?id=1' and ascii(substring((select keyid from flag limit 0,1),1,1))=33 %23

意思是： 截取 keyid 的第一个字符 ， 并转化为 ascii 与 33 进行比较，正确 就停止，不正确就继续试 34,35……

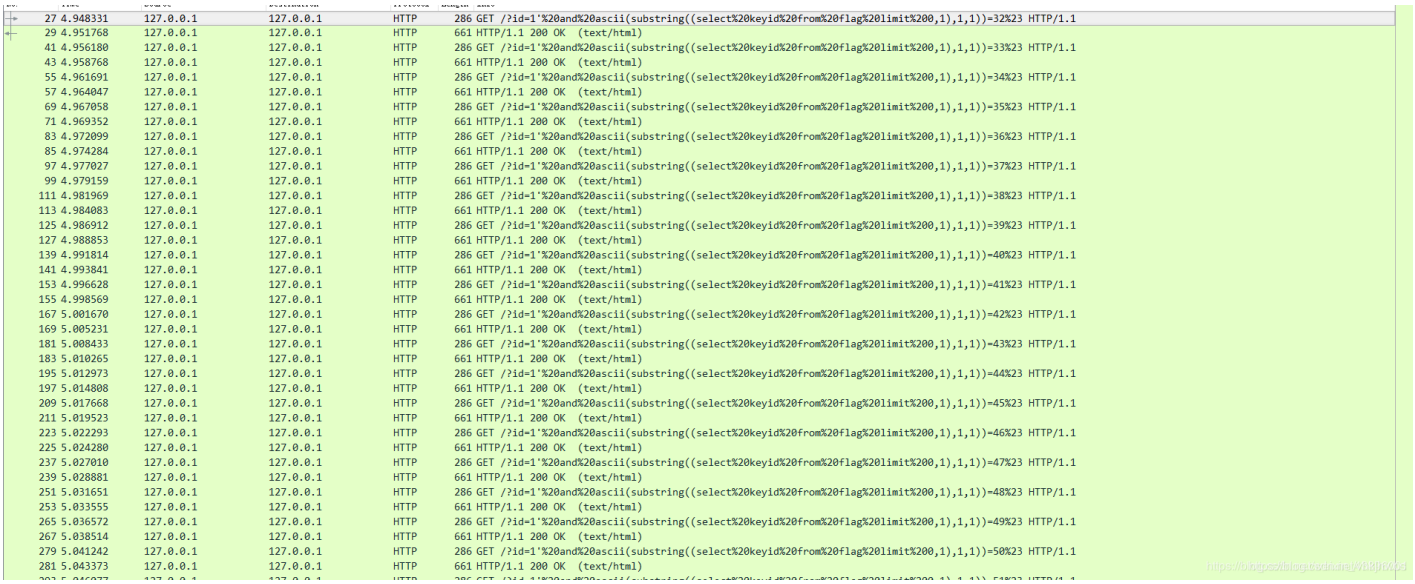
按照这个规律 得到 keyid 的 所有字符对应的 ascii ：

```
102 108 97 103 123 99 50 98 98 102 57 99 101 99 100 97 102 54 53 54 99 102 53 50 52 100 48 49 52 99 53 98 1
```

将其转化为 字符就可以了：

```
chars = "102 108 97 103 123 99 50 98 98 102 57 99 101 99 100 97 102 54 53 54 99 102 53 50 52 100 48 49 52 9
char = chars.split(" ")
for c in char :
    print(chr(int(c)),end="")
```

```
flag{c2bbf9cecdaf656cf524d014c5bf046c}
```



27	4.948331	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=32%23 HTTP/1.1
29	4.951768	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
41	4.956180	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=33%23 HTTP/1.1
43	4.958768	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
55	4.961691	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=34%23 HTTP/1.1
57	4.964047	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
69	4.967058	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=35%23 HTTP/1.1
71	4.969352	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
83	4.972099	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=36%23 HTTP/1.1
85	4.974284	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
97	4.977027	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=37%23 HTTP/1.1
99	4.979159	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
111	4.981969	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=38%23 HTTP/1.1
113	4.984083	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
125	4.986912	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=39%23 HTTP/1.1
127	4.988853	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
139	4.991814	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=40%23 HTTP/1.1
141	4.993841	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
153	4.996628	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=41%23 HTTP/1.1
155	4.998569	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
167	5.001670	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=42%23 HTTP/1.1
169	5.005231	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
181	5.008433	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=43%23 HTTP/1.1
183	5.010265	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
195	5.012973	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=44%23 HTTP/1.1
197	5.014808	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
209	5.017668	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=45%23 HTTP/1.1
211	5.019523	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
223	5.022293	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=46%23 HTTP/1.1
225	5.024280	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
237	5.027010	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=47%23 HTTP/1.1
239	5.028881	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
251	5.031651	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=48%23 HTTP/1.1
253	5.033555	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
265	5.036572	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=49%23 HTTP/1.1
267	5.038514	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)
279	5.041242	127.0.0.1	127.0.0.1	HTTP	286 GET /?id=1'%20and%20ascii(substring((select%20keyid%20from%20flag%20limit%200,1),1,1))=50%23 HTTP/1.1
281	5.043373	127.0.0.1	127.0.0.1	HTTP	661 HTTP/1.1 200 OK (text/html)

大佬的脚本直接跑：

```
import pyshark

lastt=256
flag="flag is "
capa=pyshark.FileCapture('misc.pcapng',only_summaries=True,display_filter="http.request == 1")
for arps in capa:
    now=eval(str(arps).split(" ")[7].split('=')[2].split('%')[0])
    if now < lastt & lastt <= 255:
        flag+=chr(lastt)
    lastt=now
flag+='}'
print(flag)
```

上面的看不太懂，贴一个自己写的：

（过滤 http 然后 文件 》 导出分组解析结果 》 为 CVS ,保存为 12.txt ）

```
import re
import urllib.parse

f = open(r'12.txt')
file = f.readlines()
datas = []
#urldecode
for i in file:
    datas.append(urllib.parse.unquote(i))
lines = []
#提取注入flag的语句
for i in range(0,len(datas)):
    if datas[i].find("flag limit 0,1),1,1))=32# HTTP/1.1") > 0:
        lines = datas[i:]
        break

flag = {}
macth = re.compile(r"limit 0,1\),(.*?),1\)\)=(.*?)# HTTP/1.1") #匹配我们需要的信息块
for i in range(0,len(lines),2):
    obj = macth.search(lines[i])
    if obj:
        key = int(obj.group(1)) #获取字符的位置
        value = int(obj.group(2))#获取字符的ascii值
        flag[key] = value #不断的更新，进而保留最后一个
f.close()
result = ''
for value in flag.values():
    result += chr(value)
print(result)
```

本文为基础篇，如果想进一步提升 流量分析的水平，可以参考我的另一篇文章：