****本文值得你收藏分享，就怕你错过了，就再也找不到像这样把 tcpdump 讲得直白而且特全的文章了****。

在讲解之前，有两点需要声明：

第三节到第六节里的 tcpdump 命令示例，只为了说明参数的使用，并不一定就能抓到包，如果要精准抓到你所需要的包，需要配合第五节的逻辑逻辑运算符进行组合搭配。

不同 Linux 发行版下、不同版本的 tcpdump 可能有小许差异， 本文是基于 CentOS 7.2 的 4.5.1 版本的tcpdump 进行学习的，若在你的环境中无法使用，请参考 man tcpdump 进行针对性学习。

IMG_256

## ****tcpdump 核心参数图解****

大家都知道，网络上的流量、数据包，非常的多，因此要想抓到我们所需要的数据包，就需要我们定义一个精准的过滤器，把这些目标数据包，从巨大的数据包网络中抓取出来。

所以学习抓包工具，其实就是学习如何定义过滤器的过程。

而在 tcpdump 的世界里，过滤器的实现，都是通过一个又一个的参数组合起来，一个参数不够精准，那就再加一个，直到我们能过滤掉无用的数据包，只留下我们感兴趣的数据包。

tcpdump 的参数非常的多，初学者在没有掌握 tcpdump 时，会对这个命令的众多参数产生很多的疑惑。

就比如下面这个命令，我们要通过 host 参数指定 host ip 进行过滤

$ tcpdump host 192.168.10.100

主程序 + 参数名+ 参数值  这样的组合才是我们正常认知里面命令行该有的样子。

可 tcpdump 却不走寻常路，我们居然还可以在 host 前再加一个限定词，来缩小过滤的范围？

$ tcpdump src host 192.168.10.100

从字面上理解，确实很容易理解，但是这不符合编写命令行程序的正常逻辑，导致我们会有所疑虑：

除了 src ，dst，可还有其它可以用的限定词？

src，host 应该如何理解它们，叫参数名？不合适，因为 src 明显不合适。

如果你在网上看到有关 tcpdump 的博客、教程，无一不是给你一个参数组合，告诉你这是实现了怎样的一个过滤器？这样的教学方式，很容易让你依赖别人的文章来使用 tcpdump，而不能将 tcpdump 这样神器消化，达到灵活应用，灵活搭配过滤器的效果。

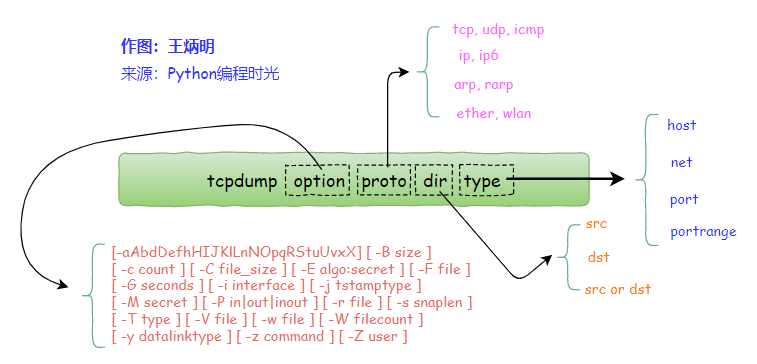
上面加了 src 本身就颠覆了我们的认知，你可知道在 src 之前还可以加更多的条件，比如 tcp, udp, icmp 等词，在你之前的基础上再过滤一层。

$ tcpdump tcp src host 192.168.10.100

这种参数的不确定性，让大多数人对 tcpdump 的学习始终无法得其精髓。

因此，在学习 tcpdump 之前，我觉得有必要要先让你知道：****tcpdump 的参数是如何组成的？这非常重要。****

为此，我画了一张图，方便你直观的理解 tcpdump 的各种参数：



option 可选参数：将在后边一一解释，对应本文 ****第四节：可选参数解析****

proto 类过滤器：根据协议进行过滤，可识别的关键词有：upd, udp, icmp, ip, ip6, arp, rarp,ether,wlan, fddi, tr, decnet

type 类过滤器：可识别的关键词有：host, net, port, portrange，这些词后边需要再接参数。

direction 类过滤器：根据数据流向进行过滤，可识别的关键字有：src, dst，同时你可以使用逻辑运算符进行组合，比如 src or dst

proto、type、direction 这三类过滤器的内容比较简单，也最常用，因此我将其放在最前面，也就是 ****第三节：常规过滤规则****一起介绍。

而 option 可选的参数非常多，有的甚至也不经常用到，因此我将其放到后面一点，也就是 ****第四节：可选参数解析****

当你看完前面六节，你对 tcpdump 的认识会上了一个台阶，至少能够满足你 80% 的使用需求。

****你一定会问了，还有 20% 呢？****

其实 tcpdump 还有一些过滤关键词，它不符合以上四种过滤规则，可能需要你单独记忆。关于这部分我会在  ****第六节：特殊过滤规则****  里进行介绍。

IMG_258

****理解 tcpdump 的输出****

### **2.1 输出内容结构**

tcpdump 输出的内容虽然多，却很规律。

这里以我随便抓取的一个 tcp 包为例来看一下

21:26:49.013621 IP 172.20.20.1.15605 > 172.20.20.2.5920: Flags [P.], seq 49:97, ack 106048, win 4723, length 48

从上面的输出来看，可以总结出：

第一列：时分秒毫秒 21:26:49.013621

第二列：网络协议 IP

第三列：发送方的ip地址+端口号，其中172.20.20.1是 ip，而15605 是端口号

第四列：箭头 >， 表示数据流向

第五列：接收方的ip地址+端口号，其中 172.20.20.2 是 ip，而5920 是端口号

第六列：冒号

第七列：数据包内容，包括Flags 标识符，seq 号，ack 号，win 窗口，数据长度 length，其中 [P.] 表示 PUSH 标志位为 1，更多标识符见下面

### **2.2 Flags 标识符**

使用 tcpdump 抓包后，会遇到的 TCP 报文 Flags，有以下几种：

[S] : SYN（开始连接）

[P] : PSH（推送数据）

[F] : FIN （结束连接）

[R] : RST（重置连接）

[.] : 没有 Flag，由于除了 SYN 包外所有的数据包都有ACK，所以一般这个标志也可表示 ACK

## 

## **IMG_259**

## ****常规过滤规则****

### **3.1 基于IP地址过滤：host**

使用 host 就可以指定 host ip 进行过滤

$ tcpdump host 192.168.10.100

数据包的 ip 可以再细分为源ip和目标ip两种

*# 根据源ip进行过滤*

$ tcpdump -i eth2 src 192.168.10.100

*# 根据目标ip进行过滤*

$ tcpdump -i eth2 dst 192.168.10.200

### **3.2 基于网段进行过滤：net**

若你的ip范围是一个网段，可以直接这样指定

$ tcpdump net 192.168.10.0/24

网段同样可以再细分为源网段和目标网段

*# 根据源网段进行过滤*

$ tcpdump src net 192.168

*# 根据目标网段进行过滤*

$ tcpdump dst net 192.168

### **3.3 基于端口进行过滤：port**

使用 port 就可以指定特定端口进行过滤

$ tcpdump port 8088

端口同样可以再细分为源端口，目标端口

*# 根据源端口进行过滤*

$ tcpdump src port 8088

*# 根据目标端口进行过滤*

$ tcpdump dst port 8088

如果你想要同时指定两个端口你可以这样写

$ tcpdump port 80 or port 8088

但也可以简写成这样

$ tcpdump port 80 or 8088

如果你的想抓取的不再是一两个端口，而是一个范围，一个一个指定就非常麻烦了，此时你可以这样指定一个端口段。

$ tcpdump portrange 8000-8080

$ tcpdump src portrange 8000-8080

$ tcpdump dst portrange 8000-8080

对于一些常见协议的默认端口，我们还可以直接使用协议名，而不用具体的端口号

比如 http  == 80，https == 443 等

$ tcpdump tcp port http

### **3.4 基于协议进行过滤：proto**

常见的网络协议有：tcp, udp, icmp, http, ip,ipv6 等

若你只想查看 icmp 的包，可以直接这样写

$ tcpdump icmp

protocol 可选值：ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl,  moprc,  iso,  stp, ipx,  or  netbeui

### **3.5 基本IP协议的版本进行过滤**

当你想查看 tcp 的包，你也许会这样子写

$ tcpdump tcp

这样子写也没问题，就是不够精准，为什么这么说呢？

ip 根据版本的不同，可以再细分为 IPv4 和 IPv6 两种，如果你只指定了 tcp，这两种其实都会包含在内。

那有什么办法，能够将 IPv4 和 IPv6 区分开来呢？

很简单，如果是 IPv4 的 tcp 包 ，就这样写（友情提示：数字 6 表示的是 tcp 在ip报文中的编号。）

$ tcpdump 'ip proto tcp'

*# or*

$ tcpdump ip proto 6

*# or*

$ tcpdump 'ip protochain tcp'

*# or*

$ tcpdump ip protochain 6

而如果是 IPv6 的 tcp 包 ，就这样写

$ tcpdump 'ip6 proto tcp'

*# or*

$ tcpdump ip6 proto 6

*# or*

$ tcpdump 'ip6 protochain tcp'

*# or*

$ tcpdump ip6 protochain 6

关于上面这几个命令示例，有两点需要注意：

跟在 proto 和 protochain 后面的如果是 tcp, udp, icmp ，那么过滤器需要用引号包含，这是因为 tcp,udp, icmp 是 tcpdump 的关键字。

跟在ip 和 ip6 关键字后面的 proto 和 protochain 是两个新面孔，看起来用法类似，它们是否等价，又有什么区别呢？

关于第二点，网络上没有找到很具体的答案，我只能通过 man tcpdump 的提示， 给出自己的个人猜测，但不保证正确。

proto 后面跟的 <protocol> 的关键词是固定的，只能是 ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl,  moprc,  iso,  stp, ipx,  or  netbeui 这里面的其中一个。

而 protochain 后面跟的 protocol 要求就没有那么严格，它可以是任意词，只要 tcpdump 的 IP 报文头部里的 protocol 字段为 <protocol> 就能匹配上。

理论上来讲，下面两种写法效果是一样的

$ tcpdump 'ip && tcp'

$ tcpdump 'ip proto tcp'

同样的，这两种写法也是一样的

$ tcpdump 'ip6 && tcp'

$ tcpdump 'ip6 proto tcp'

## 

IMG_260

## ****可选参数解析****

### **4.1 设置不解析域名提升速度**

-n：不把ip转化成域名，直接显示  ip，避免执行 DNS lookups 的过程，速度会快很多

-nn：不把协议和端口号转化成名字，速度也会快很多。

-N：不打印出host 的域名部分.。比如,，如果设置了此选现，tcpdump 将会打印'nic' 而不是 'nic.ddn.mil'.

### **4.2 过滤结果输出到文件**

使用 tcpdump 工具抓到包后，往往需要再借助其他的工具进行分析，比如常见的 wireshark 。

而要使用wireshark ，我们得将 tcpdump 抓到的包数据生成到文件中，最后再使用 wireshark 打开它即可。

使用 -w 参数后接一个以 .pcap 后缀命令的文件名，就可以将 tcpdump 抓到的数据保存到文件中。

$ tcpdump icmp -w icmp.pcap

### **4.3 从文件中读取包数据**

使用 -w 是写入数据到文件，而使用 -r 是从文件中读取数据。

读取后，我们照样可以使用上述的过滤器语法进行过滤分析。

$ tcpdump icmp -r all.pcap

### **4.4 控制详细内容的输出**

-v：产生详细的输出. 比如包的TTL，id标识，数据包长度，以及IP包的一些选项。同时它还会打开一些附加的包完整性检测，比如对IP或ICMP包头部的校验和。

-vv：产生比-v更详细的输出. 比如NFS回应包中的附加域将会被打印, SMB数据包也会被完全解码。（摘自网络，目前我还未使用过）

-vvv：产生比-vv更详细的输出。比如 telent 时所使用的SB, SE 选项将会被打印, 如果telnet同时使用的是图形界面，其相应的图形选项将会以16进制的方式打印出来（摘自网络，目前我还未使用过）

### **4.5 控制时间的显示**

-t：在每行的输出中不输出时间

-tt：在每行的输出中会输出时间戳

-ttt：输出每两行打印的时间间隔(以毫秒为单位)

-tttt：在每行打印的时间戳之前添加日期的打印（此种选项，输出的时间最直观）

### **4.6 显示数据包的头部**

-x：以16进制的形式打印每个包的头部数据（但不包括数据链路层的头部）

-xx：以16进制的形式打印每个包的头部数据（包括数据链路层的头部）

-X：以16进制和 ASCII码形式打印出每个包的数据(但不包括连接层的头部)，这在分析一些新协议的数据包很方便。

-XX：以16进制和 ASCII码形式打印出每个包的数据(包括连接层的头部)，这在分析一些新协议的数据包很方便。

### **4.7 过滤指定网卡的数据包**

-i：指定要过滤的网卡接口，如果要查看所有网卡，可以 -i any

### **4.8 过滤特定流向的数据包**

-Q：选择是入方向还是出方向的数据包，可选项有：in, out, inout，也可以使用  --direction=[direction] 这种写法

### **4.9 其他常用的一些参数**

-A：以ASCII码方式显示每一个数据包(不显示链路层头部信息). 在抓取包含网页数据的数据包时, 可方便查看数据

-l : 基于行的输出，便于你保存查看，或者交给其它工具分析

-q : 简洁地打印输出。即打印很少的协议相关信息, 从而输出行都比较简短.

-c : 捕获 count 个包 tcpdump 就退出

-s :  tcpdump 默认只会截取前 96 字节的内容，要想截取所有的报文内容，可以使用 -s number， number 就是你要截取的报文字节数，如果是 0 的话，表示截取报文全部内容。

-S : 使用绝对序列号，而不是相对序列号

-C：file-size，tcpdump 在把原始数据包直接保存到文件中之前, 检查此文件大小是否超过file-size. 如果超过了, 将关闭此文件,另创一个文件继续用于原始数据包的记录. 新创建的文件名与-w 选项指定的文件名一致, 但文件名后多了一个数字.该数字会从1开始随着新创建文件的增多而增加. file-size的单位是百万字节(nt: 这里指1,000,000个字节,并非1,048,576个字节, 后者是以1024字节为1k, 1024k字节为1M计算所得, 即1M=1024 ＊ 1024 ＝ 1,048,576)

-F：使用file 文件作为过滤条件表达式的输入, 此时命令行上的输入将被忽略.

### **4.10 对输出内容进行控制的参数**

-D : 显示所有可用网络接口的列表

-e : 每行的打印输出中将包括数据包的数据链路层头部信息

-E : 揭秘IPSEC数据

-L ：列出指定网络接口所支持的数据链路层的类型后退出

-Z：后接用户名，在抓包时会受到权限的限制。如果以root用户启动tcpdump，tcpdump将会有超级用户权限。

-d：打印出易读的包匹配码

-dd：以C语言的形式打印出包匹配码.

-ddd：以十进制数的形式打印出包匹配码

## 

## **IMG_261**

## ****过滤规则组合****

有编程基础的同学，对于下面三个逻辑运算符应该不陌生了吧

and：所有的条件都需要满足，也可以表示为 &&

or：只要有一个条件满足就可以，也可以表示为 ||

not：取反，也可以使用 !

举个例子，我想需要抓一个来自10.5.2.3，发往任意主机的3389端口的包

$ tcpdump src 10.5.2.3 and dst port 3389

当你在使用多个过滤器进行组合时，有可能需要用到括号，而括号在 shell 中是特殊符号，因为你需要使用引号将其包含。例子如下：

$ tcpdump 'src 10.0.2.4 and (dst port 3389 or 22)'

而在单个过滤器里，常常会判断一条件是否成立，这时候，就要使用下面两个符号

=：判断二者相等

==：判断二者相等

!=：判断二者不相等

当你使用这两个符号时，tcpdump 还提供了一些关键字的接口来方便我们进行判断，比如

if：表示网卡接口名、

proc：表示进程名

pid：表示进程 id

svc：表示 service class

dir：表示方向，in 和 out

eproc：表示 effective process name

epid：表示 effective process ID

比如我现在要过滤来自进程名为 nc 发出的流经 en0 网卡的数据包，或者不流经 en0 的入方向数据包，可以这样子写

$ tcpdump "( if=en0 and proc =nc ) || (if != en0 and dir=in)"

## 

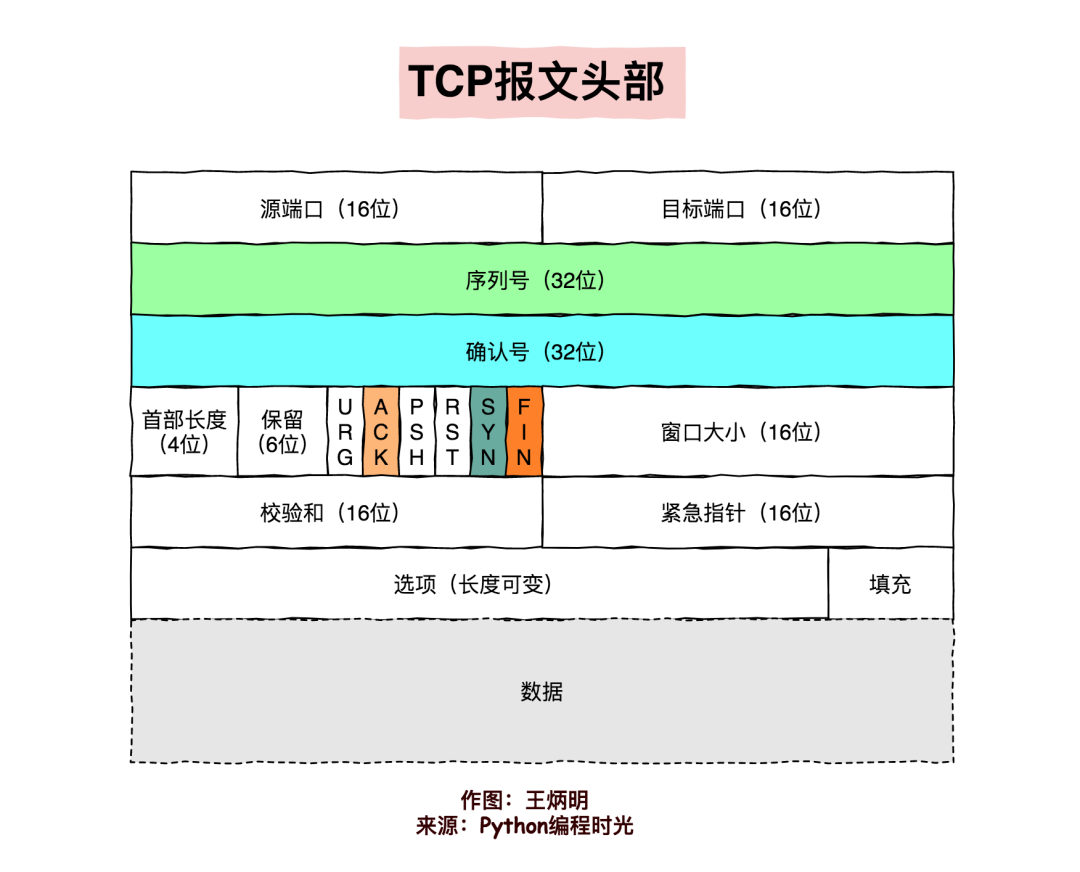
## 

## **IMG_262**

## ****特殊过滤规则****

### **5.1 根据 tcpflags 进行过滤**

通过[上一篇文章](http://mp.weixin.qq.com/s?__biz=MzIzMzMzOTI3Nw==&chksm=e8867256dff1fb40c9f47bafd0e87a9237c5a9ebf33c8a3d0a598276b496d29cdaa3fbff8d26&idx=1&mid=2247488180&scene=21&sn=09526224732ebfcccb52847f27298c70" \l "wechat_redirect)，我们知道了 tcp 的首部有一个标志位。



TCP 报文首部

tcpdump 支持我们根据数据包的标志位进行过滤

proto [ expr:size ]

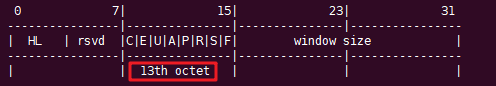
proto：可以是熟知的协议之一（如ip，arp，tcp，udp，icmp，ipv6）

expr：可以是数值，也可以是一个表达式，表示与指定的协议头开始处的字节偏移量。

size：是可选的，表示从字节偏移量开始取的字节数量。

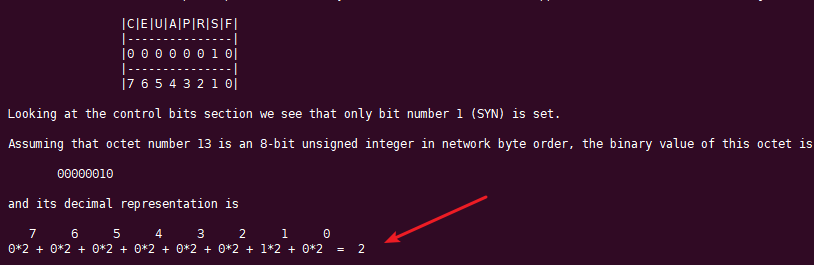
接下来，我将举几个例子，让人明白它的写法，不过在那之前，有几个点需要你明白，这在后面的例子中会用到：

****1、****tcpflags 可以理解为是一个别名常量，相当于 13，它代表着与指定的协议头开头相关的字节偏移量，也就是标志位，所以 tcp[tcpflags] 等价于 tcp[13] ，对应下图中的报文位置。



****2、****tcp-fin, tcp-syn, tcp-rst, tcp-push, tcp-ack, tcp-urg 这些同样可以理解为别名常量，分别代表 1，2，4，8，16，32，64。这些数字是如何计算出来的呢？

以 tcp-syn 为例，你可以参照下面这张图，计算出来的值 是就是 2



由于数字不好记忆，所以一般使用这样的“别名常量”表示。

因此当下面这个表达式成立时，就代表这个包是一个 syn 包。

tcp[tcpflags] == tcp-syn

要抓取特定数据包，方法有很多种。

下面以最常见的 syn包为例，演示一下如何用 tcpdump 抓取到 syn 包，而其他的类型的包也是同样的道理。

据我总结，主要有三种写法：

1、第一种写法：使用数字表示偏移量

$ tcpdump -i eth0 "tcp[13] & 2 != 0"

2、第二种写法：使用别名常量表示偏移量

$ tcpdump -i eth0 "tcp[tcpflags] & tcp-syn != 0"

3、第三种写法：使用混合写法

$ tcpdump -i eth0 "tcp[tcpflags] & 2 != 0"

*# or*

$ tcpdump -i eth0 "tcp[13] & tcp-syn != 0"

如果我想同时捕获多种类型的包呢，比如 syn + ack 包

1、第一种写法

$ tcpdump -i eth0 'tcp[13] == 2 or tcp[13] == 16'

2、第二种写法

$ tcpdump -i eth0 'tcp[tcpflags] == tcp-syn or tcp[tcpflags] == tcp-ack'

3、第三种写法

$ tcpdump -i eth0 "tcp[tcpflags] & (tcp-syn|tcp-ack) != 0"

4、第四种写法：注意这里是 单个等号，而不是像上面一样两个等号，18（syn+ack） = 2（syn） + 16（ack）

$ tcpdump -i eth0 'tcp[13] = 18'

*# or*

$ tcpdump -i eth0 'tcp[tcpflags] = 18'

tcp 中有 类似 tcp-syn 的别名常量，其他协议也是有的，比如 icmp 协议，可以使用的别名常量有

icmp-echoreply, icmp-unreach, icmp-sourcequench,

icmp-redirect, icmp-echo, icmp-routeradvert,

icmp-routersolicit, icmp-timx-ceed, icmp-paramprob,

icmp-tstamp, icmp-tstampreply,icmp-ireq,

icmp-ireqreply, icmp-maskreq, icmp-maskreply

### **5.2  基于包大小进行过滤**

若你想查看指定大小的数据包，也是可以的

$ tcpdump less 32

$ tcpdump greater 64

$ tcpdump <= 128

### **5.3 根据 mac 地址进行过滤**

例子如下，其中 ehost 是记录在 /etc/ethers 里的 name

$ tcpdump ether host [ehost]

$ tcpdump ether dst    [ehost]

$ tcpdump ether src    [ehost]

### **5.4 过滤通过指定网关的数据包**

$ tcpdump gateway [host]

### **5.5 过滤广播/多播数据包**

$ tcpdump ether broadcast

$ tcpdump ether multicast

$ tcpdump ip broadcast

$ tcpdump ip multicast

$ tcpdump ip6 multicast

## 

IMG_266

## ****如何抓取到更精准的包？****

先给你抛出一个问题：如果我只想抓取 HTTP 的 POST 请求该如何写呢？

如果只学习了上面的内容，恐怕你还是无法写法满足这个抓取需求的过滤器。

在学习之前，我先给出答案，然后再剖析一下，这个过滤器是如何生效的，居然能让我们对包内的内容进行判断。

$ tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4]'

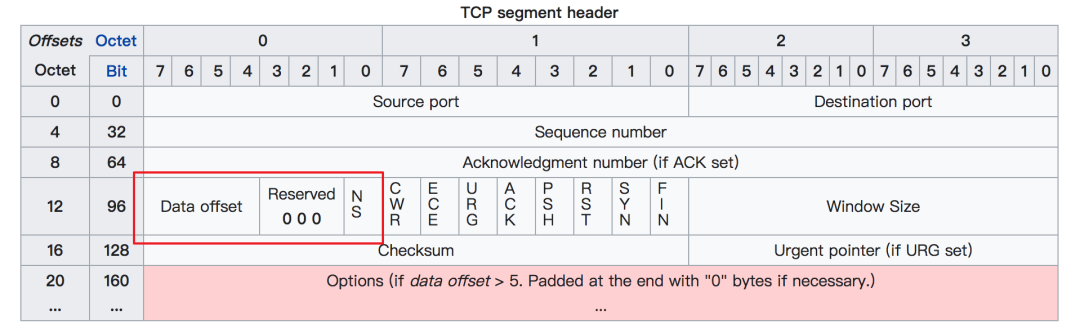
命令里的可选参数，在前面的内容里已经详细讲过了。这里不再细讲。

本节的重点是引号里的内容，看起来很复杂的样子。

将它逐一分解，我们只要先理解了下面几种用法，就能明白

tcp[n]：表示 tcp 报文里 第 n 个字节

tcp[n:c]：表示 tcp 报文里从第n个字节开始取 c 个字节，tcp[12:1] 表示从报文的第12个字节（因为有第0个字节，所以这里的12其实表示的是13）开始算起取一个字节，也就是 8 个bit。查看 tcp 的报文首部结构，可以得知这 8 个bit 其实就是下图中的红框圈起来的位置，而在这里我们只要前面 4个bit，也就是实际数据在整个报文首部中的偏移量。

* 

&：是位运算里的 and 操作符，比如 0011 & 0010 = 0010

>>：是位运算里的右移操作，比如 0111 >> 2 = 0011

0xf0：是 10 进制的 240 的 16 进制表示，但对于位操作来说，10进制和16进制都将毫无意义，我们需要的是二进制，将其转换成二进制后是：11110000，这个数有什么特点呢？前面个 4bit 全部是 1，后面4个bit全部是0，往后看你就知道这个特点有什么用了。

分解完后，再慢慢合并起来看

1、tcp[12:1] & 0xf0 其实并不直观，但是我们将它换一种写法，就好看多了，假设 tcp 报文中的 第12 个字节是这样组成的 10110000，那么这个表达式就可以变成 10110110 && 11110000 = 10110000，得到了 10110000 后，再进入下一步。

2、tcp[12:1] & 0xf0) >> 2 ：如果你不理解 tcp 报文首部里的数据偏移，请先点击这个前往我的[上一篇文章](http://mp.weixin.qq.com/s?__biz=MzIzMzMzOTI3Nw==&chksm=e8867256dff1fb40c9f47bafd0e87a9237c5a9ebf33c8a3d0a598276b496d29cdaa3fbff8d26&idx=1&mid=2247488180&scene=21&sn=09526224732ebfcccb52847f27298c70" \l "wechat_redirect)，搞懂数据偏移的意义，否则我保证你这里会绝对会听懵了。

tcp[12:1] & 0xf0) >> 2 这个表达式实际是 (tcp[12:1] & 0xf0) >> 4 ) << 2 的简写形式。所以要搞懂 tcp[12:1] & 0xf0) >> 2 只要理解了(tcp[12:1] & 0xf0) >> 4 ) << 2  就行了 。

从上一步我们算出了 tcp[12:1] & 0xf0  的值其实是一个字节，也就是 8 个bit，但是你再回去看下上面的 tcp 报文首部结构图，表示数据偏移量的只有 4个bit，也就是说 上面得到的值 10110000，前面 4 位（1011）才是正确的偏移量，那么为了得到 1011，只需要将 10110000 右移4位即可，也就是 tcp[12:1] & 0xf0) >> 4，至此我们是不是已经得出了实际数据的正确位置呢，很遗憾还没有，前一篇文章里我们讲到 Data Offset 的单位是 4个字节，因为要将 1011 乘以 4才可以，除以4在位运算中相当于左移2位，也就是 <<2，与前面的 >>4 结合起来一起算的话，最终的运算可以简化为 >>2

至此，我们终于得出了实际数据开始的位置是 tcp[12:1] & 0xf0) >> 2 （单位是字节）。

找到了数据的起点后，可别忘了我们的目的是从数据中打到 HTTP 请求的方法，是 GET 呢 还是 POST ，或者是其他的？

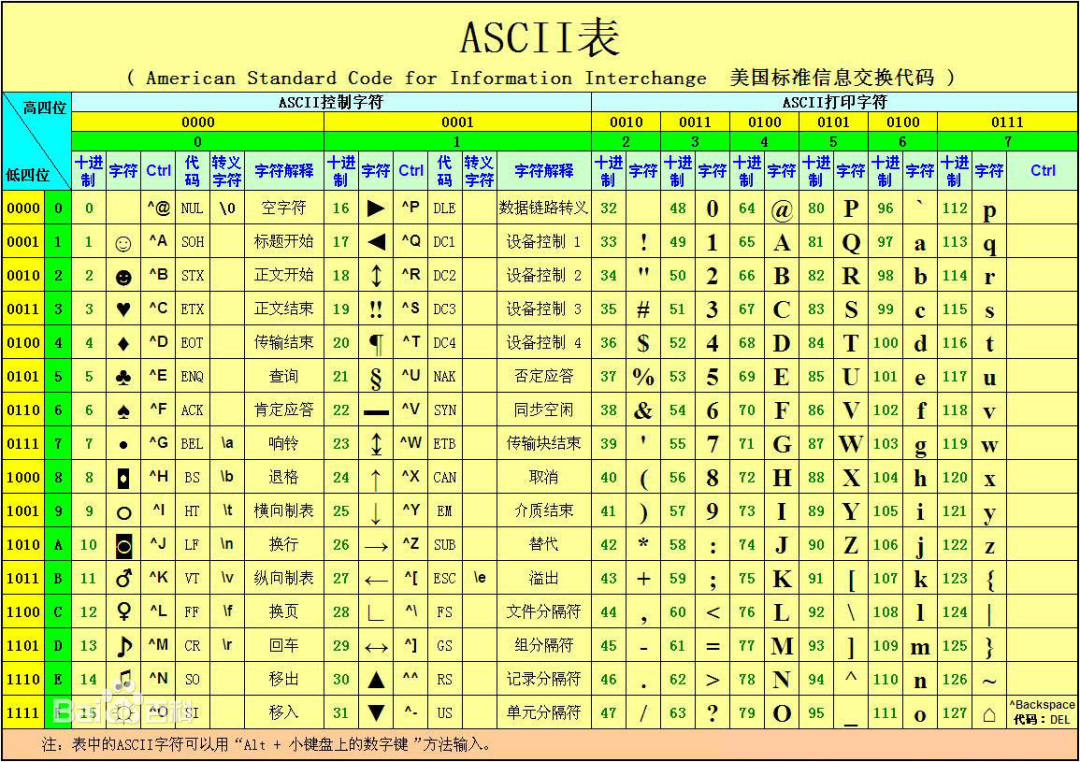
有了上面的经验，我们自然懂得使用 tcp[((tcp[12:1] & 0xf0) >> 2):4] 从数据开始的位置再取出四个字节，然后将结果与 GET （注意 GET最后还有个空格）的 16进制写法（也就是 0x47455420）进行比对。

0x47   -->   71    -->  G

0x45   -->   69    -->  E

0x54   -->   84    -->  T

0x20   -->   32    -->  空格



如果相等，则该表达式为True，tcpdump 认为这就是我们所需要抓的数据包，将其输出到我们的终端屏幕上。

## **IMG_269**

## ****抓包实战应用例子****

### **以下例子摘自：*https://fuckcloudnative.io/posts/tcpdump-examples/***

### **8.1 提取 HTTP 的 User-Agent**

从 HTTP 请求头中提取 HTTP 用户代理：

$ tcpdump -nn -A -s1500 -l | grep "User-Agent:"

通过 egrep 可以同时提取用户代理和主机名（或其他头文件）：

$ tcpdump -nn -A -s1500 -l | egrep -i 'User-Agent:|Host:'

### **8.2 抓取 HTTP GET 和 POST 请求**

抓取 HTTP GET 请求包：

$ tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420'

*# or*

$ tcpdump -vvAls0 | grep 'GET'

可以抓取 HTTP POST 请求包：

$ tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x504f5354'

*# or*

$ tcpdump -vvAls0 | grep 'POST'

注意：该方法不能保证抓取到 HTTP POST 有效数据流量，因为一个 POST 请求会被分割为多个 TCP 数据包。

### **8.3 找出发包数最多的 IP**

找出一段时间内发包最多的 IP，或者从一堆报文中找出发包最多的 IP，可以使用下面的命令：

$ tcpdump -nnn -t -c 200 | cut -f 1,2,3,4 -d '.' | sort | uniq -c | sort -nr | head -n 20

****cut -f 1,2,3,4 -d '.'**** : 以 . 为分隔符，打印出每行的前四列。即 IP 地址。

****sort | uniq -c**** : 排序并计数

****sort -nr**** : 按照数值大小逆向排序

### **8.4 抓取 DNS 请求和响应**

DNS 的默认端口是 53，因此可以通过端口进行过滤

$ tcpdump -i any -s0 port 53

### **8.5 切割 pcap 文件**

当抓取大量数据并写入文件时，可以自动切割为多个大小相同的文件。例如，下面的命令表示每 3600 秒创建一个新文件 capture-(hour).pcap，每个文件大小不超过 200\*1000000 字节：

$ tcpdump  -w /tmp/capture-%H.pcap -G 3600 -C 200

这些文件的命名为 capture-{1-24}.pcap，24 小时之后，之前的文件就会被覆盖。

### **8.6 提取 HTTP POST 请求中的密码**

从 HTTP POST 请求中提取密码和主机名：

$ tcpdump -s 0 -A -n -l | egrep -i "POST /|pwd=|passwd=|password=|Host:"

### **8.7 提取 HTTP 请求的 URL**

提取 HTTP 请求的主机名和路径：

$ tcpdump -s 0 -v -n -l | egrep -i "POST /|GET /|Host:"

### **8.8 抓取 HTTP 有效数据包**

抓取 80 端口的 HTTP 有效数据包，排除 TCP 连接建立过程的数据包（SYN / FIN / ACK）：

$ tcpdump 'tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'

### **8.9 结合 Wireshark 进行分析**

通常 Wireshark（或 tshark）比 tcpdump 更容易分析应用层协议。一般的做法是在远程服务器上先使用 tcpdump 抓取数据并写入文件，然后再将文件拷贝到本地工作站上用 Wireshark 分析。

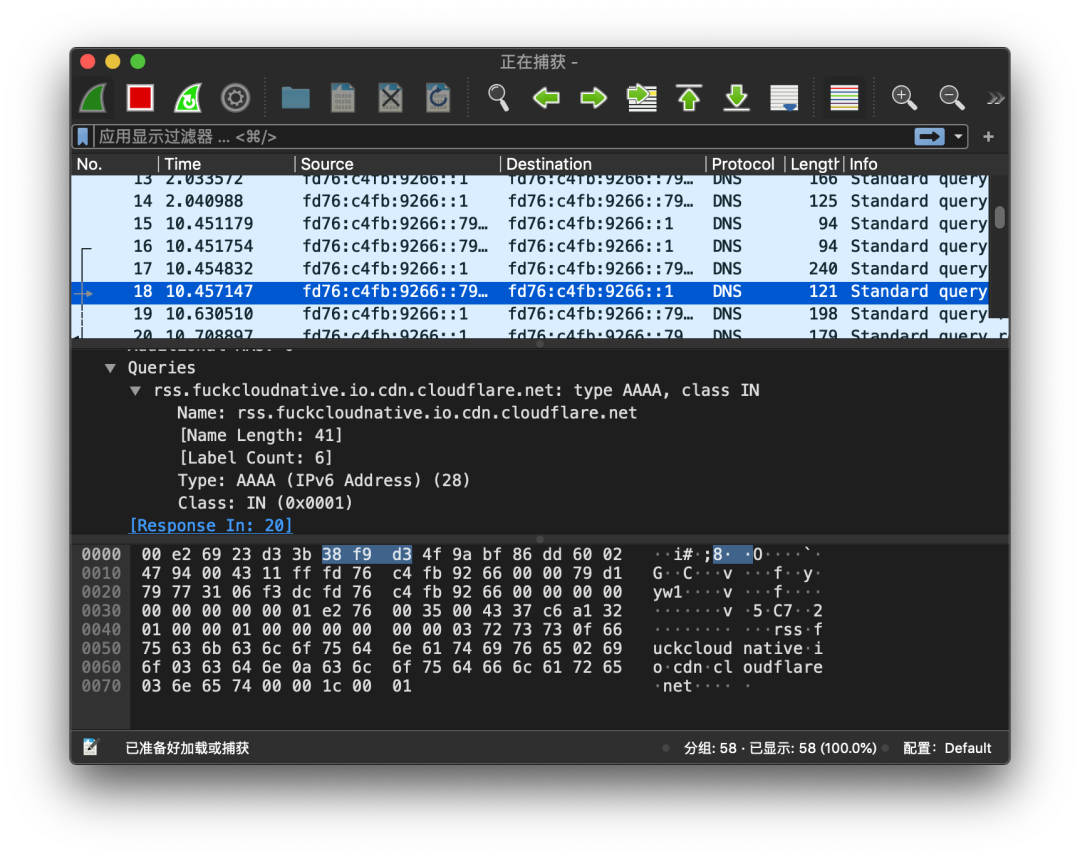
还有一种更高效的方法，可以通过 ssh 连接将抓取到的数据实时发送给 Wireshark 进行分析。以 MacOS 系统为例，可以通过 brew cask install wireshark 来安装，然后通过下面的命令来分析：

$ ssh root@remotesystem 'tcpdump -s0 -c 1000 -nn -w - not port 22' | /Applications/Wireshark.app/Contents/MacOS/Wireshark -k -i -

例如，如果想分析 DNS 协议，可以使用下面的命令：

$ ssh root@remotesystem 'tcpdump -s0 -c 1000 -nn -w - port 53' | /Applications/Wireshark.app/Contents/MacOS/Wireshark -k -i -

抓取到的数据：



-c 选项用来限制抓取数据的大小。如果不限制大小，就只能通过 ctrl-c 来停止抓取，这样一来不仅关闭了 tcpdump，也关闭了 wireshark。