**/\* WireShark 过滤语法 \*/**

**1.过滤IP，如来源IP或者目标IP等于某个IP**  
例子:  
ip.src eq 192.168.1.107 or ip.dst eq 192.168.1.107  
或者  
ip.addr eq 192.168.1.107 // 都能显示来源IP和目标IP  
  
**2.过滤端口**  
例子:  
tcp.port eq 80 // 不管端口是来源的还是目标的都显示  
tcp.port == 80  
tcp.port eq 2722  
tcp.port eq 80 or udp.port eq 80  
tcp.dstport == 80 // 只显tcp协议的目标端口80  
tcp.srcport == 80 // 只显tcp协议的来源端口80  
  
udp.port eq 15000  
  
过滤端口范围  
tcp.port >= 1 and tcp.port <= 80  
  
**3.过滤协议**  
例子:  
tcp  
udp  
arp  
icmp  
http  
smtp  
ftp  
dns  
msnms  
ip  
ssl  
oicq  
bootp  
等等  
  
排除arp包，如!arp   或者   not arp

**4.过滤MAC**  
 以太网头过滤  
eth.dst == A0:00:00:04:C5:84 // 过滤目标mac  
eth.src eq A0:00:00:04:C5:84 // 过滤来源mac  
eth.dst==A0:00:00:04:C5:84  
eth.dst==A0-00-00-04-C5-84  
eth.addr eq A0:00:00:04:C5:84 // 过滤来源MAC和目标MAC都等于A0:00:00:04:C5:84的  
  
less than 小于 < lt   
小于等于 le  
  
等于 eq  
大于 gt  
大于等于 ge  
不等 ne

**5.包长度过滤**  
例子:  
udp.length == 26 这个长度是指udp本身固定长度8加上udp下面那块数据包之和  
tcp.len >= 7   指的是ip数据包(tcp下面那块数据),不包括tcp本身  
ip.len == 94 除了以太网头固定长度14,其它都算是ip.len,即从ip本身到最后  
frame.len == 119 整个数据包长度,从eth开始到最后  
  
eth ---> ip or arp ---> tcp or udp ---> data  
  
**6.http模式过滤**  
例子:  
http.request.method == "GET"  
http.request.method == "POST"  
http.request.uri == "/img/logo-edu.gif"  
http contains "GET"  
http contains "HTTP/1."  
  
// GET包  
http.request.method == "GET" && http contains "Host: "  
http.request.method == "GET" && http contains "User-Agent: "  
// POST包  
http.request.method == "POST" && http contains "Host: "  
http.request.method == "POST" && http contains "User-Agent: "  
// 响应包  
http contains "HTTP/1.1 200 OK" && http contains "Content-Type: "  
http contains "HTTP/1.0 200 OK" && http contains "Content-Type: "  
一定包含如下  
Content-Type:

**7.TCP参数过滤**  
tcp.flags 显示包含TCP标志的封包。  
tcp.flags.syn == 0x02     显示包含TCP SYN标志的封包。  
tcp.window\_size == 0 && tcp.flags.reset != 1  
  
**8.过滤内容**  
  
tcp[20]表示从20开始，取1个字符  
tcp[20:]表示从20开始，取1个字符以上  
tcp[20:8]表示从20开始，取8个字符  
tcp[offset,n]  
  
udp[8:3]==81:60:03 // 偏移8个bytes,再取3个数，是否与==后面的数据相等？  
udp[8:1]==32   如果我猜的没有错的话，应该是udp[offset:截取个数]=nValue  
eth.addr[0:3]==00:06:5B  
  
例子:  
判断upd下面那块数据包前三个是否等于0x20 0x21 0x22  
我们都知道udp固定长度为8  
udp[8:3]==20:21:22  
  
判断tcp那块数据包前三个是否等于0x20 0x21 0x22  
tcp一般情况下，长度为20,但也有不是20的时候  
tcp[8:3]==20:21:22  
如果想得到最准确的，应该先知道tcp长度  
  
matches(匹配)和contains(包含某字符串)语法  
ip.src==192.168.1.107 and udp[8:5] matches "\\x02\\x12\\x21\\x00\\x22"  
ip.src==192.168.1.107 and udp contains 02:12:21:00:22  
ip.src==192.168.1.107 and tcp contains "GET"  
udp contains 7c:7c:7d:7d 匹配payload中含有0x7c7c7d7d的UDP数据包，不一定是从第一字节匹配。  
  
例子:  
得到本地qq登陆数据包(判断条件是第一个包==0x02,第四和第五个包等于0x00x22,最后一个包等于0x03)  
0x02 xx xx 0x00 0x22 ... 0x03  
正确  
oicq and udp[8:] matches "^\\x02[\\x00-\\xff][\\x00-\\xff]\\x00\\x22[\\x00-\\xff]+\\x03$"  
oicq and udp[8:] matches "^\\x02[\\x00-\\xff]{2}\\x00\\x22[\\x00-\\xff]+\\x03$" // 登陆包  
oicq and (udp[8:] matches "^\\x02[\\x00-\\xff]{2}\\x03$" or tcp[8:] matches "^\\x02[\\x00-\\xff]{2}\\x03$")  
oicq and (udp[8:] matches "^\\x02[\\x00-\\xff]{2}\\x00\\x22[\\x00-\\xff]+\\x03$" or tcp[20:] matches "^\\x02[\\x00-\\xff]{2}\\x00\\x22[\\x00-\\xff]+\\x03$")  
  
不单单是00:22才有QQ号码,其它的包也有,要满足下面条件(tcp也有，但没有做):  
oicq and udp[8:] matches "^\\x02[\\x00-\\xff]+\\x03$" and !(udp[11:2]==00:00) and !(udp[11:2]==00:80)  
oicq and udp[8:] matches "^\\x02[\\x00-\\xff]+\\x03$" and !(udp[11:2]==00:00) and !(udp[15:4]==00:00:00:00)  
说明:  
udp[15:4]==00:00:00:00 表示QQ号码为空  
udp[11:2]==00:00 表示命令编号为00:00  
udp[11:2]==00:80 表示命令编号为00:80  
当命令编号为00:80时，QQ号码为00:00:00:00  
  
得到msn登陆成功账号(判断条件是"USR 7 OK ",即前三个等于USR，再通过两个0x20，就到OK,OK后面是一个字符0x20,后面就是mail了)  
USR xx OK mail@hotmail.com  
正确  
msnms and tcp and ip.addr==192.168.1.107 and tcp[20:] matches "^USR\\x20[\\x30-\\x39]+\\x20OK\\x20[\\x00-\\xff]+"

**9.dns模式过滤**  
  
**10.DHCP**  
以寻找伪造DHCP服务器为例，介绍Wireshark的用法。在显示过滤器中加入过滤规则，  
显示所有非来自DHCP服务器并且bootp.type==0x02（Offer/Ack）的信息：  
bootp.type==0x02 and not ip.src==192.168.1.1  
  
**11.msn**  
msnms && tcp[23:1] == 20 // 第四个是0x20的msn数据包  
msnms && tcp[20:1] >= 41 && tcp[20:1] <= 5A && tcp[21:1] >= 41 && tcp[21:1] <= 5A && tcp[22:1] >= 41 && tcp[22:1] <= 5A  
msnms && tcp[20:3]=="USR" // 找到命令编码是USR的数据包  
msnms && tcp[20:3]=="MSG" // 找到命令编码是MSG的数据包  
tcp.port == 1863 || tcp.port == 80  
  
如何判断数据包是含有命令编码的MSN数据包?  
1)端口为1863或者80,如:tcp.port == 1863 || tcp.port == 80  
2)数据这段前三个是大写字母,如:  
tcp[20:1] >= 41 && tcp[20:1] <= 5A && tcp[21:1] >= 41 && tcp[21:1] <= 5A && tcp[22:1] >= 41 && tcp[22:1] <= 5A  
3)第四个为0x20,如:tcp[23:1] == 20  
4)msn是属于TCP协议的,如tcp  
  
MSN Messenger 协议分析  
<http://blog.csdn.net/Hopping/archive/2008/11/13/3292257.aspx>  
  
MSN 协议分析  
<http://blog.csdn.net/lzyzuixin/archive/2009/03/13/3986597.aspx>  
  
更详细的说明  
<<wireshark过滤表达式实例介绍>>  
<http://www.csna.cn/viewthread.php?tid=14614>  
  
Wireshark 主界面的操作菜单中英对比  
<http://www.csna.cn/viewthread.php?tid=9645&extra=page%3D1>  
  
又一款好的网络分析软件  
"科来网络分析系统"  
  
学习Ethereal/Wireshark网站  
<http://www.csna.cn/index.php>

###########################################################################

补充：[超详细解析wireshark捕获过滤器语法 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/272685665)

[Wireshark 过滤器语法 - 粪球 - 博客园 (cnblogs.com)](https://www.cnblogs.com/corsacsherry/p/4517161.html)

[Wireshark过滤器写法总结 - willingtolove - 博客园 (cnblogs.com)](https://www.cnblogs.com/willingtolove/p/12519490.html#%E6%98%BE%E7%A4%BA%E8%BF%87%E6%BB%A4%E5%99%A8%E5%86%99%E6%B3%95)

[(4条消息) wireshark过滤语法总结\_cumirror的博客-CSDN博客\_wireshark](https://blog.csdn.net/cumirror/article/details/7054496)

[wireshark使用教程及过滤语法总结——血泪史的汇聚 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/28090794)

[(4条消息) wireshark抓包新手使用教程\_zterrorblade的博客-CSDN博客\_wireshark抓包新手使用教程](https://blog.csdn.net/zterrorblade/article/details/120916749)

[wireshark网络抓包详解 - 大数据老司机 - 博客园 (cnblogs.com)](https://www.cnblogs.com/liugp/p/16343201.html)

**1、wireshark基本的语法字符**  
  
\d          0-9的数字  
\D          \d的补集（以所以字符为全集，下同），即所有非数字的字符  
\w          单词字符，指大小写字母、0-9的数字、下划线  
\W          \w的补集  
\s          空白字符，包括换行符\n、回车符\r、制表符\t、垂直制表符\v、换页符\f  
\S          \s的补集  
.          除换行符\n外的任意字符。 在Perl中，“.”可以匹配新行符的模式被称作“单行模式”  
.\*       匹配任意文本，不包括回车(\n)? 。 而，[0x00-0xff]\*        匹配任意文本,包括\n  
[…]         匹配[]内所列出的所有字符  
[^…]        匹配非[]内所列出的字符  
  
----------------------------------------------------------------------------------------  
**2、定位字符**  所代表的是一个虚的字符，它代表一个位置，你也可以直观地认为“定位字符”所代表的是某个字符与字符间的那个微小间隙。  
  
^          表示其后的字符必须位于字符串的开始处  
$          表示其前面的字符必须位于字符串的结束处  
\b          匹配一个单词的边界  
\B          匹配一个非单词的边界  
  
----------------------------------------------------------------------------------------  
**3、重复描述字符**  
  
{n}          匹配前面的字符n次  
{n,}          匹配前面的字符n次或多于n次  
{n,m}          匹配前面的字符n到m次  
?          匹配前面的字符0或1次  
+          匹配前面的字符1次或多于1次  
\*          匹配前面的字符0次或式于0次  
  
----------------------------------------------------------------------------------------  
**4、and or 匹配**  
  
and 符号 并  
or  符号 或  
例如：  
tcp and tcp.port==80  
tcp or udp  
  
----------------------------------------------------------------------------------------  
**5、wireshark过滤匹配表达式实例**  
  
5.1、搜索按条件过滤udp的数据段payload（数字8是表示udp头部有8个字节，数据部分从第9个字节开始udp[8:]）  
udp[8]==14        (14是十六进制0x14)匹配payload第一个字节0x14的UDP数据包  
udp[8:2]==14:05 可以udp[8:2]==1405，且只支持2个字节连续，三个以上须使用冒号：分隔表示十六进制。 (相当于 udp[8]==14 and udp[9]==05,1405是0x1405)  
udp[8:3]==22:00:f7 但是不可以udp[8:3]==2200f7  
udp[8:4]==00:04:00:2a，匹配payload的前4个字节0x0004002a  
而udp contains 7c:7c:7d:7d 匹配payload中含有0x7c7c7d7d的UDP数据包，不一定是从第一字节匹配。  
udp[8:4] matches "\\x14\\x05\\x07\\x18"  
udp[8:] matches "^\\x14\\x05\\x07\\x18\\x14"  
  
5.2、搜索按条件过滤tcp的数据段payload（数字20是表示tcp头部有20个字节，数据部分从第21个字节开始tcp[20:]）  
tcp[20:] matches "^GET [ -~]\*HTTP/1.1\\x0d\\x0a"  
等同http matches "^GET [ -~]\*HTTP/1.1\\x0d\\x0a"  
  
tcp[20:] matches "^GET (.\*?)HTTP/1.1\\x0d\\x0a"  
tcp[20:] matches "^GET (.\*?)HTTP/1.1\\x0d\\x0a[\\x00-\\xff]\*Host: (.\*?)pplive(.\*?)\\x0d\\x0a"  
tcp[20:] matches "^GET (.\*?)HTTP/1.1\\x0d\\x0a[\\x00-\\xff]\*Host: "  
tcp[20:] matches "^POST / HTTP/1.1\\x0d\\x0a[\\x00-\\xff]\*\\x0d\\x0aConnection: Keep-Alive\\x0d\\x0a\\x0d\\x0a"  
  
检测SMB头的smb标记，指明smb标记从tcp头部第24byte的位置开始匹配。  
tcp[24:4] == ff:53:4d:42  
  
检测SMB头的smb标记，tcp的数据包含十六进制ff:53:4d:42，从tcp头部开始搜索此数据。  
tcp contains ff:53:4d:42  
tcp matches "\\xff\\x53\\x4d\\x42"  
  
检测tcp含有十六进制01:bd,从tcp头部开始搜索此数据。  
tcp matches "\\x01\\xbd"  
  
检测MS08067的RPC请求路径  
tcp[179:13] == 00:5c:00:2e:00:2e:00:5c:00:2e:00:2e:00  
                  \      .     .     \     .     .  
5.3、其他  
http.request.uri matches ".gif$" 匹配过滤HTTP的请求URI中含有".gif"字符串，并且以.gif结尾（4个字节）的http请求数据包（$是正则表达式中的结尾表示符）  
注意区别：http.request.uri contains ".gif$" 与此不同，contains是包含字符串".gif$"（5个字节）。匹配过滤HTTP的请求URI中含有".gif$"字符串的http请求数据包（这里$是字符，不是结尾符）  
  
eth.addr[0:3]==00:1e:4f 搜索过滤MAC地址前3个字节是0x001e4f的数据包。