文件相关

前置技能

- 了解常见的编码

  能够对文件中出现的一些编码进行解码，并且对一些特殊的编码（Base64、十六进制、二进制等）有一定的敏感度，对其进行转换并得到最终的 flag。
- 能够利用脚本语言（Python 等）去操作二进制数据
- 熟知常见文件的文件格式，尤其是各类 文件头、协议、结构等
- 灵活运用常见的工具

常用工具

010 Editor（需购买）

SweetScape 010 Editor 是一个全新的十六进位文件编辑器，它有别于传统的十六进位编辑器在于它可用「范本」来解析二进位文件，从而让你读懂和编辑它。它还可用来比较一切可视的二进位文件。

利用它的模板功能可以非常轻松的观察文件内部的具体结构并且依此快速更改内容。

HxD

功能简洁的16进制编辑器，免费。

file 命令

`file` 命令根据文件头（魔法字节）去识别一个文件的文件类型。

```
root in ~/Desktop/tmp λ file flag
flag: PNG image data, 450 x 450, 8-bit grayscale, non-interlaced
```

strings 命令

打印文件中可打印的字符，经常用来发现文件中的一些提示信息或是一些特殊的编码信息，常常用来发现题目的突破口。

- 可以配合 `grep` 命令探测指定信息

  ```
  strings test|grep -i flag
  ```
- 也可以配合 `-o` 参数获取所有 ASCII 字符偏移

```
root in ~/Desktop/tmp λ strings -o flag|head
    14 IHDR
    45 gAMA
    64  cHRM
    141 bKGD
    157 tIME
    202 IDATx
    223 NFdVK3
    361 |;*-
    410 Ge%<W
    431 5duX@%
```

binwalk 命令

binwalk 本是一个固件的分析工具，比赛中常用来发现多个文件粘合再在一起的情况。根据文件头去识别一个文件中夹杂的其他文件，有时也会存在误报率（尤其是对 Pcap 流量包等文件时）。

```
root in ~/Desktop/ binwalk flag

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 450 x 450, 8-bit grayscale, non-interlaced
134           0x86            Zlib compressed data, best compression
25683         0x6453          Zip archive data, at least v2.0 to extract, compressed size: 675, uncompressed size: 1159, name: readme.txt
26398         0x671E          Zip archive data, at least v2.0 to extract, compressed size: 430849, uncompressed size: 1027984, name: trid
457387        0x6FAAB         End of Zip archive
```

配合 `-e` 参数可以进行自动化提取。

也可以结合 `dd` 命令进行手动切割。

```
root in ~/Desktop/tmp λ dd if=flag of=1.zip bs=1 skip=25683
431726+0 records in
431726+0 records out
431726 bytes (432 kB, 422 KiB) copied, 0.900973 s, 479 kB/s
```

foremost 命令

可以非常方便的分解由多个文件拼成的，和binwalk一样在面对流量包文件时有很大误报。

文件头

在window下一个文件的类型取决于文件头的规定，默认打开方式取决于后缀名。

一个文件的真实类型需要用16进制编辑器打开，去查看文件头，然后再通过附件的文件头大全去查才能获得真实类型。

所以拿到一个文件的第一步最好是先使用HxD或者notepad++打开，先搜索一下flag，然后再查看文件头决定下一步怎么做。

图片文件

常见jpg，png，gif等

元数据隐写

元数据（Metadata），又称中介数据、中继数据，为描述数据的数据（Data about data），主要是描述数据属性（property）的信息，用来支持如指示存储位置、历史数据、资源查找、文件记录等功能。

元数据中隐藏信息在比赛中是最基本的一种手法，通常用来隐藏一些关键的 Hint 信息或者一些重要的比如password 等信息。

这类元数据你可以 `右键 --> 属性` 去查看, 也可以通过 `strings` 命令去查看，一般来说，一些隐藏的信息（奇怪的字符串）常常出现在头部或者尾部。

接下来介绍一个 `identify` 命令，这个命令是用来获取一个或多个图像文件的格式和特性。

`-format` 用来指定显示的信息，灵活使用它的 `-format` 参数可以给解题带来不少方便。[format 各个参数具体意义]

JPG文件

- JPEG 是有损压缩格式，将像素信息用 JPEG 保存成文件再读取出来，其中某些像素值会有少许变化。在保存时有个质量参数可在 0 至 100 之间选择，参数越大图片就越保真，但图片的体积也就越大。一般情况下选择 70 或 80 就足够了
- JPEG 没有透明度信息

JPG 基本数据结构为两大类型：「段」和经过压缩编码的图像数据。

名 称	字节数	数据	说明
段 标识	1	FF	每个新段的开始标识
段类型	1		类型编码（称作标记码）
段长 度	2		包括段内容和段长度本身, 不包括段标识和段类型
段内容	2		≤65533 字节

- 有些段没有长度描述也没有内容，只有段标识和段类型。文件头和文件尾均属于这种段。
- 段与段之间无论有多少 `FF` 都是合法的，这些 `FF` 称为「填充字节」，必须被忽略掉。

Stegdetect](https://github.com/redNixon/stegdetect)

通过统计分析技术评估 JPEG 文件的 DCT 频率系数的隐写工具, 可以检测到通过 JSteg、JPHide、OutGuess、Invisible Secrets、F5、appendX 和 Camouflage 等这些隐写工具隐藏的信息，并且还具有基于字典暴力破解密码方法提取通过 Jphide、outguess 和 jsteg-shell 方式嵌入的隐藏信息。

```
-q 仅显示可能包含隐藏内容的图像。
-n 启用检查JPEG文件头功能，以降低误报率。如果启用，所有带有批注区域的文件将被视为没有被嵌入信息。如果JPEG文件的JFIF标识符中的版本号不是1.1，则禁用OutGuess检测。
-s 修改检测算法的敏感度，该值的默认值为1。检测结果的匹配度与检测算法的敏感度成正比，算法敏感度的值越大，检测出的可疑文件包含敏感信息的可能性越大。
-d 打印带行号的调试信息。
-t 设置要检测哪些隐写工具（默认检测jopi），可设置的选项如下：
j 检测图像中的信息是否是用jsteg嵌入的。
o 检测图像中的信息是否是用outguess嵌入的。
p 检测图像中的信息是否是用jphide嵌入的。
i 检测图像中的信息是否是用invisible secrets嵌入的。
```

JPHS

JPEG 图像的信息隐藏软件 JPHS，它是由 Allan Latham 开发设计实现在 Windows 和 Linux 系统平台针对有损压缩 JPEG 文件进行信息加密隐藏和探测提取的工具。软件里面主要包含了两个程序 JPHIDE 和 JPSEEK。JPHIDE 程序主要是实现将信息文件加密隐藏到 JPEG 图像功能，而 JPSEEK 程序主要实现从用 JPHIDE 程序加密隐藏得到的 JPEG 图像探测提取信息文件，Windows 版本的 JPHS 里的 JPHSWIN 程序具有图形化操作界面且具备 JPHIDE 和 JPSEEK 的功能。

jpg一般不会出太复杂的隐写，上述仅供最后测试，一般情况下测试元数据和组合文件即可

PNG文件

文件头：89 50 4E 47 0D 0A 1A 0A + 数据块 + 数据块 + 数据块.....

数据块 CHUNk

PNG 定义了两种类型的数据块，一种是称为关键数据块（critical chunk），这是标准的数据块，另一种叫做辅助数据块（ancillary chunks），这是可选的数据块。关键数据块定义了 4 个标准数据块，每个 PNG 文件都必须包含它们，PNG 读写软件也都必须要支持这些数据块。

数据块符号	数据块名称	多数据块	可选否	位置限制
IHDR	文件头数据块	否	否	第一块
cHRM	基色和白色点数据块	否	是	在 PLTE 和 IDAT 之前
gAMA	图像γ数据块	否	是	在 PLTE 和 IDAT 之前
sBIT	样本有效位数据块	否	是	在 PLTE 和 IDAT 之前
PLTE	调色板数据块	否	是	在 IDAT 之前
bKGD	背景颜色数据块	否	是	在 PLTE 之后 IDAT 之前
hIST	图像直方图数据块	否	是	在 PLTE 之后 IDAT 之前
tRNS	图像透明数据块	否	是	在 PLTE 之后 IDAT 之前
oFFs	(专用公共数据块)	否	是	在 IDAT 之前
pHYs	物理像素尺寸数据块	否	是	在 IDAT 之前
sCAL	(专用公共数据块)	否	是	在 IDAT 之前
IDAT	图像数据块	是	否	与其他 IDAT 连续
tIME	图像最后修改时间数据块	否	是	无限制
tEXt	文本信息数据块	是	是	无限制
zTXt	压缩文本数据块	是	是	无限制
fRAc	(专用公共数据块)	是	是	无限制
gIFg	(专用公共数据块)	是	是	无限制
gIFt	(专用公共数据块)	是	是	无限制
gIFx	(专用公共数据块)	是	是	无限制
IEND	图像结束数据	否	否	最后一个数据块

对于每个数据块都有着统一的数据结构，每个数据块由 4 个部分组成

名称	字节数	说明
Length（长度）	4 字节	指定数据块中数据域的长度，其长度不超过（231 - 1）字节
Chunk Type Code（数据块类型码）	4 字节	数据块类型码由 ASCII 字母（A - Z 和 a - z）组成
Chunk Data（数据块数据）	可变长度	存储按照 Chunk Type Code 指定的数据
CRC（循环冗余检测）	4 字节	存储用来检测是否有错误的循环冗余码

CRC（Cyclic Redundancy Check）域中的值是对 Chunk Type Code 域和 Chunk Data 域中的数据进行计算得到的。

IHDR

文件头数据块 IHDR（Header Chunk）：它包含有 PNG 文件中存储的图像数据的基本信息，由 13 字节组成，并要作为第一个数据块出现在 PNG 数据流中，而且一个 PNG 数据流中只能有一个文件头数据块

其中我们关注的是前 8 字节的内容

域的名称	字节数	说明
Width	4 bytes	图像宽度，以像素为单位
Height	4 bytes	图像高度，以像素为单位

我们经常会去更改一张图片的高度或者宽度使得一张图片显示不完整从而达到隐藏信息的目的。

这种图片一般在 Kali 中是打不开的，提示 `IHDR CRC error`，而 Windows 10 自带的图片查看器能够打开，就提醒了我们 IHDR 块被人为的篡改过了，从而尝试修改图片的高度或者宽度发现隐藏的字符串。

有一些情况随意改大高度会导致无法打开，所以需要通过爆破crc来得到真实高度修改，附上一个脚本

```
import os
import binascii
import struct


misc = open("height.png","rb").read()

for i in range(1024):
    data = misc[12:16] + struct.pack('>i',i)+ misc[20:29]
    crc32 = binascii.crc32(data) & 0xffffffff
    if crc32 == 0x932f8a6b:
        print i
```

0x932f8a6b就是偏移量1d左右开始的crc码

IDAT

图像数据块 IDAT（image data chunk）：它存储实际的数据，在数据流中可包含多个连续顺序的图像数据块。

- 储存图像像数数据
- 在数据流中可包含多个连续顺序的图像数据块
- 采用 LZ77 算法的派生算法进行压缩
- 可以用 zlib 解压缩

值得注意的是，IDAT 块只有当上一个块充满时，才会继续一个新的块。

用 `pngcheck` 去查看此 PNG 文件

```
λ .\pngcheck.exe -v sctf.png
File: sctf.png (1421461 bytes)
  chunk IHDR at offset 0x0000c, length 13
    1000 x 562 image, 32-bit RGB+alpha, non-interlaced
  chunk sRGB at offset 0x00025, length 1
```

```
    rendering intent = perceptual
  chunk gAMA at offset 0x00032, length 4: 0.45455
  chunk pHYs at offset 0x00042, length 9: 3780x3780 pixels/meter (96 dpi)
  chunk IDAT at offset 0x00057, length 65445
    zlib: deflated, 32K window, fast compression
  chunk IDAT at offset 0x10008, length 65524
...
  chunk IDAT at offset 0x150008, length 45027
  chunk IDAT at offset 0x15aff7, length 138
  chunk IEND at offset 0x15b08d, length 0
No errors detected in sctf.png (28 chunks, 36.8% compression).
```

可以看到，正常的块的 length 是在 65524 的时候就满了，而倒数第二个 IDAT 块长度是 45027，最后一个长度是 138，很明显最后一个 IDAT 块是有问题的，因为他本来应该并入到倒数第二个未满的块里.

利用 `python zlib` 解压多余 IDAT 块的内容，此时注意剔除 **长度、数据块类型及末尾的 CRC 校验值。**

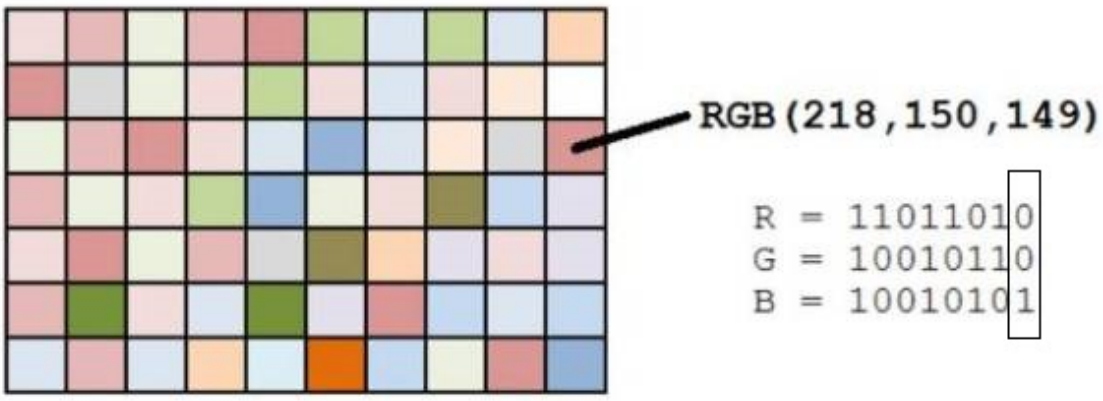
```
import zlib
import binascii
IDAT = "789...667".decode('hex')
result = binascii.hexlify(zlib.decompress(IDAT))
print result
```

LSB

LSB 全称 Least Significant Bit，最低有效位。PNG 文件中的图像像数一般是由 RGB 三原色（红绿蓝）组成，每一种颜色占用 8 位，取值范围为 `0x00` 至 `0xFF`，即有 256 种颜色，一共包含了 256 的 3 次方的颜色，即 16777216 种颜色。

而人类的眼睛可以区分约 1000 万种不同的颜色，意味着人类的眼睛无法区分余下的颜色大约有 6777216 种。

LSB 隐写就是修改 RGB 颜色分量的最低二进制位（LSB），每个颜色会有 8 bit，LSB 隐写就是修改了像数中的最低的 1 bit，而人类的眼睛不会注意到这前后的变化，每个像素可以携带 3 比特的信息。



如果是要寻找这种 LSB 隐藏痕迹的话，推荐使用 Stegsolve 、 **zsteg**，可以很方便解决LSB隐写。

GIF文件

文件头一般为 47 49 46 38 39 61 转换成ascii为GIF89a

空间轴

由于 GIF 的动态特性，由一帧帧的图片构成，所以每一帧的图片，多帧图片间的结合，都成了隐藏信息的一种载体。

convert xxx.gif xxx.png 或者工具包里的gif分离工具即可

时间轴

GIF 文件每一帧间的时间间隔也可以作为信息隐藏的载体。

通过`identify`可以的打印出每一帧的时间间隔

```
identify -format "%s %T \n" xxx.gif
```

比较少见

音频文件

直接听

听上去像摩斯电码的直接先测一下`.-`来回试一下。

MP3文件

MP3 隐写主要是使用 Mp3Stego 工具进行隐写，其使用方法如下

```
encode -E hidden_text.txt -P pass svega.wav svega_stego.mp3
decode -X -P pass svega_stego.mp3
```

一般方案：先用strings命令或者编辑器打开，查找flag，key，pwd，password之类的关键字，如果有密码之类的东西就用那个密码去解mp3stego。

波形隐写

原理

通常来说，波形方向的题，在观察到异常后，使用相关软件（Audacity, Adobe Audition 等）观察波形规律，将波形进一步转化为 01 字符串等，从而提取转化出最终的 flag。

特点：**波形隐写的题往往声音特别明显**，听声音异常即可使用

频谱隐写

原理

音频中的频谱隐写是将字符串隐藏在频谱中，此类音频通常会有一个较明显的特征，听起来是一段杂音或者比较刺耳。

使用Audacity去查看频谱

LSB音频隐写

类似于图片隐写中的 LSB 隐写，音频中也有对应的 LSB 隐写。主要可以使用 Silenteye 工具。

压缩文件

ZIP文件

文件结构

`ZIP` 文件主要由三部分构成，分别为

压缩源文件数据区	核心目录	目录结束
local file header + file data + data descriptor	central directory	end of central directory record

- 压缩源文件数据区中每一个压缩的源文件或目录都是一条记录，其中
  - `local file header`：文件头用于标识该文件的开始，记录了该压缩文件的信息，这里的文件头标识由固定值 `50 4B 03 04` 开头，也是 `ZIP` 的文件头的重要标志
  - `file data`：文件数据记录了相应压缩文件的数据
  - `data descriptor`：数据描述符用于标识该文件压缩结束，该结构只有在相应的 `local file header` 中通用标记字段的第 `3 bit` 设为 `1` 时才会出现，紧接在压缩文件源数据后
- `Central directory` 核心目录
- 记录了压缩文件的目录信息，在这个数据区中每一条纪录对应在压缩源文件数据区中的一条数据。

Offset	Bytes	Description	译

Offset	Bytes	Description	译
0	4	Central directory file header signature = 0x04034b50	核心目录文件 header 标识 =（0x04034b50）
4	2	Version made by	压缩所用的 pkware 版本
6	2	Version needed to extract (minimum)	解压所需 pkware 的最低版本
8	2	General purpose bit flag	通用位标记伪加密
10	2	Compression method	压缩方法
12	2	File last modification time	文件最后修改时间
14	2	File last modification date	文件最后修改日期
16	4	CRC-32	CRC-32 校验码
20	4	Compressed size	压缩后的大小
24	4	Uncompressed size	未压缩的大小
28	2	File name length (n)	文件名长度
30	2	Extra field length (m)	扩展域长度
32	2	File comment length (k)	文件注释长度
34	2	Disk number where file starts	文件开始位置的磁盘编号
36	2	Internal file attributes	内部文件属性
38	4	External file attributes	外部文件属性
42	4	relative offset of local header	本地文件头的相对位移
46	n	File name	目录文件名
46+n	m	Extra field	扩展域
46+n+m	k	File comment	文件注释内容

- `End of central directory record(EOCD)` 目录结束标识
- 目录结束标识存在于整个归档包的结尾，用于标记压缩的目录数据的结束。每个压缩文件必须有且只有一个 `EOCD` 记录。

加密文件

伪加密

处理伪加密的方法：

- `16` 进制下修改通用位标记
- `binwalk -e` 无视伪加密
- 在 `Mac OS` 及部分 `Linux`(如 `Kali`) 系统中，可以直接打开伪加密的 `ZIP` 压缩包
- 检测伪加密的小工具 `ZipCenOp.jar`

- 有时候用 `WinRar` 的修复功能（此方法有时有奇效，不仅针对伪加密）

真加密

windows 下常常使用 **ARCHPR** 来处理真加密的压缩文件

一般有以下几种情况：

1. 有压缩文件包里的文件的明文 -----> 已知明文攻击
2. 有密码相关线索 -----> 掩码，字典爆破
3. 完全没有相关线索，但是压缩包文件较小（小于8byte）----->CRC爆破

已知明文攻击

原理：

- 压缩文件的压缩工具，比如 `2345` 好压， `WinRAR` ， `7z` 。 `zip` 版本号等，可以通过文件属性了解。如果是 `Linux` 平台，用 `zipinfo -v` 可以查看一个 `zip` 包的详细信息，包括加密算法等
- 知道压缩包里某个文件的部分连续内容 (至少 `12` 字节)

如果你已经知道加密文件的部分内容，比如在某个网站上发现了它的 `readme.txt` 文件，你就可以开始尝试破解了。

首先，将这个明文文件打包成 `zip` 包，比如将 `readme.txt` 打包成 `readme.zip` 。

打包完成后，需要确认二者采用的压缩算法相同。

字典爆破

直接生成字典，然后使用**ARCHPR**去爆破。

CRC32爆破

原理：

`CRC` 本身是「冗余校验码」的意思， `CRC32` 则表示会产生一个 `32 bit` (`8` 位十六进制数) 的校验值。由于 `CRC32` 产生校验值时源数据块的每一个 `bit` (位) 都参与了计算，所以数据块中即使只有一位发生了变化，也会得到不同的 `CRC32` 值。

`CRC32` 校验码出现在很多文件中比如 `png` 文件，同样 `zip` 中也有 `CRC32` 校验码。值得注意的是 `zip` 中的 `CRC32` 是未加密文件的校验值。

这也就导致了基于 `CRC32` 的攻击手法。

- 文件内内容很少 (一般比赛中大多为 `4` 字节左右)
- 加密的密码很长

我们不去爆破压缩包的密码，而是直接去直接爆破源文件的内容 (一般都是可见的字符串)，从而获取想要的信息。

使用工具包中的脚本爆破即可。

RAR文件

RAR 文件主要由标记块，压缩文件头块，文件头块，结尾块组成。

其每一块大致分为以下几个字段：

名称	大小	描述
HEAD_CRC	2	全部块或块部分的 CRC
HEAD_TYPE	1	块类型
HEAD_FLAGS	2	阻止标志
HEAD_SIZE	2	块大小
ADD_SIZE	4	可选字段 - 添加块大小

Rar 压缩包的文件头为 `0x 52 61 72 21 1A 07 00`。

紧跟着文件头（0x526172211A0700）的是标记块（MARK_HEAD），其后还有文件头（File Header）。

名称	大小	描述
HEAD_CRC	2	CRC of fields from HEAD_TYPE to FILEATTR and file name
HEAD_TYPE	1	Header Type: 0x74
HEAD_FLAGS	2	Bit Flags (Please see ‘Bit Flags for File in Archive’ table for all possibilities)（伪加密）
HEAD_SIZE	2	File header full size including file name and comments
PACK_SIZE	4	Compressed file size
UNP_SIZE	4	Uncompressed file size
HOST_OS	1	Operating system used for archiving (See the ‘Operating System Indicators’ table for the flags used)
FILE_CRC	4	File CRC

名称	大小	描述
FTIME	4	Date and time in standard MS DOS format
UNP_VER	1	RAR version needed to extract file (Version number is encoded as 10 * Major version + minor version.)
METHOD	1	Packing method (Please see 'Packing Method' table for all possibilities
NAME_SIZE	2	File name size
ATTR	4	File attributes
HIGH_PACK_SIZ	4	High 4 bytes of 64-bit value of compressed file size. Optional value, presents only if bit 0x100 in HEAD_FLAGS is set.
HIGH_UNP_SIZE	4	High 4 bytes of 64-bit value of uncompressed file size. Optional value, presents only if bit 0x100 in HEAD_FLAGS is set.
FILE_NAME	NAME_SIZE bytes	File name - string of NAME_SIZE bytes size
SALT	8	present if (HEAD_FLAGS & 0x400) != 0
EXT_TIME	variable size	present if (HEAD_FLAGS & 0x1000) != 0

每个 RAR 文件的结尾快（Terminator）都是固定的。

Field Name	Size (bytes)	Possibilities
HEAD_CRC	2	Always 0x3DC4
HEAD_TYPE	1	Header type: 0x7b
HEAD_FLAGS	2	Always 0x4000
HEAD_SIZE	2	Block size = 0x0007

攻击方案与ZIP类似，直接参考上面zip部分即可

流量包分析

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

流量包修复

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 Online
- PcapFix

协议分析

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。

显示过滤器基础语法 Protocol String1 String2 ComparisonOperator Value LogicalOperations 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

运算符	说明
==	等于
!=	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于

Logical Expressions

运算符	说明
and , &&	与
or , ||	或
! , not	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

数据提取

单文件提取 一般步骤：

1. 追踪流
2. 复制原始数据
3. 在16进制编辑器中创建

多数据提取：

常用方法

tshark -r xxx.pcap –Y xxx -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 -T fields 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)