# linux

基于Fedora18

## 发行版本

### Fedora

#### 18

##### 18的变化

fedora每次发布新版本都会发生巨变，本次变化--

/etc/sysconfig 已停止使用

/etc/sysconfig/clock 由 /etc/localtime 代替

/etc/sysconfig/i18n 已由 /etc/locale.conf 代替

/etc/sysconfig/keyboard 变为 /etc/vconsole.conf

主机名配置由 /etc/sysconfig/network 移至 /etc/hostname

Fedora 18 支持 UEFI 安全引导。即在启用了安全引导的计算机上能够引导 Fedora.有工具可让管理员创建自定义证书，以签署本地环境下对 GRUB 和内核的修改。

通过默认启用 rngd 改善了随机数生成。

使用 SELinux 和 virt-sandbox 后，服务可在安全沙盒中运行，即使是 root 用户也是如此。virt-sandbox-service 软件包将创建挂载点和 libvirt 容器。

现在有了 libseccomp 库，提供给应用程序一个简单的方式来减少误用内核 kernel syscall 过滤器带来的伤害。得益于此，虚拟机软件比如 QEMU/KVM 已经开始使用 libseccomp 了。

usermode,一个为无特权用户提供特权访问的封装器，已经被更好的 polkit 所取代。

Fedora 18 将 Kerberos 证书缓存的标准位置变更到 /run/user/$UID 来增强安全性并简化用于 NFSv4 的缓存的定位。Fedora 的 Kerberos 支持将允许用户维护适用于多个身份的证书，且允许GSSAPI 客户端代码可以依据目标服务和主机名自动选择证书。

无特权用户能否使用 halt（8）， poweroff（8） 和 reboot（8） 命令现在由 polkit 控制。参看 /usr/share/polkit-1/actions/org.freedesktop.login1.policy 中的行为描述。将不再使用放置在 /etc/pam.d/{halt,poweroff,reboot} 中的 PAM 配置文件。里面的内容，如果有的话，也将被忽略。

Fedora 18 新增 FedFS,该程序用来为多个文件服务器提供一致的命名空间。

/tmp 默认会位于 tempfs.较大的临时文件会存储在 /var/tmp.这样可以减少磁盘产生的输入输出。从而延长了 SSD 寿命，节约了功耗并提高了 /tmp 文件系统的性能。

Fedora 18 为在服务器中常见的 GPU 提供了 Kernel

Mode Setting （KMS） 驱动支持。这些 GPU 的用户现在可以享用由 KMS 驱动带来的额外功能，包括在虚拟终端中的强化图形功能。新的 KMS 驱动支持包括基于 ServerEngines 的AST和 MGA 芯片组。

X.org 服务器得到了重构来支持 GPU 热插拔。具体来说，这将允许 Fedora 更好的支持通过 USB 连接的外置图形设备，常见于各种现代系统和笔记本底座中。用户不再需要重启 X.org 服务来识别这些设备了。

切换显示管理器不再需要编辑 /etc/sysconfig/desktop 文件。只需使用命令：systemctl

enable --force displaymanager.service 就可以启用您所选择的已安装的显示管理器。

面向 Fedora 18 的 MATE 桌面 首次发布。它为用户带来了期望已久的经典、直观且易用的桌面体验。Fedora 对 MATE>的支持通过 #fedora-mate 和 #mate 提供。这两个 irc 频道位于irc.freenode.net 或者通过 Fedora 的 bugzilla.

简体中文默认引擎变为 ibus-libpinyin

### ubuntu

## 常用命令

### bg/fg/ctrl+z

总结：

（1） CTRL+Z停止进程并放入后台

（2） jobs 显示当前暂停的进程

（3） bg %N 使第N个任务在后台运行（%前有空格）

（4） fg %N 使第N个任务在前台运行

默认bg，fg不带%N时表示对最后一个进程操作！

### du 查看文件目录大小和文件夹包含的文件数

统计总数大小

**du -sh filename**（其实我们经常用du -sh \*，显示当前目录下所有的文件及其大小，如果要排序再在后面加上 | sort -n）

du -sm \* | sort -n //统计当前目录大小 并按大小 排序

du -sk \* | sort -n

du -sk \* | grep filename //查看一个文件夹的大小

du -m | cut -d "/" -f 2 //看第二个/ 字符前的文字

查看此文件夹有多少文件 /\*/\*/\* 有多少文件

du filename/

du filename/\*/\*/\* |wc -l

40752

解释：

wc [-lmw]

参数说明：

-l :多少行

-m:多少字符

-w:多少字

或者也可以打开Nautilus窗口（文件夹首选项），点击“编辑”-“首选项”，切换到“显示”标签，改变“图标标题”下面的树形列表的内容为“大小，修改日期、类型”。设置立即生效。现在在文件图标的下面会显示文件的大小。

PS：

df -hl 查看磁盘剩余空间

du -sm文件夹

返回该文件夹总M数

更多功能查看：

df --help

du --help

du --help

用法：du [选项]... [文件]...

总结每个<文件>的磁盘用量，目录则取总用量。

长选项必须用的参数在使用短选项时也是必须的。

-a, --all write counts for all files, not just directories

--apparent-size print apparent sizes, rather than disk usage; although

the apparent size is usually smaller, it may be

larger due to holes in (`sparse') files, internal

fragmentation, indirect blocks, and the like

-B, --block-size=SIZE use SIZE-byte blocks

-b, --bytes equivalent to `--apparent-size --block-size=1'

-c, --total produce a grand total

-D, --dereference-args dereference FILEs that are symbolic links

-H like --si, but also evokes a warning; will soon

change to be equivalent to --dereference-args (-D)

-h, --human-readable print sizes in human readable format (e.g., 1K 234M 2G)

--si like -h, but use powers of 1000 not 1024

-k like --block-size=1K

-l, --count-links count sizes many times if hard linked

-L, --dereference dereference all symbolic links

-P, --no-dereference don't follow any symbolic links (this is the default)

-0, --null end each output line with 0 byte rather than newline

-S, --separate-dirs do not include size of subdirectories

-s, --summarize display only a total for each argument

-x, --one-file-system skip directories on different filesystems

-X FILE, --exclude-from=FILE Exclude files that match any pattern in FILE.

--exclude=PATTERN Exclude files that match PATTERN.

--max-depth=N print the total for a directory (or file, with --all)

only if it is N or fewer levels below the command

line argument; --max-depth=0 is the same as

--summarize

--help 显示此帮助信息并离开

--version 显示版本信息并离开

eg: du -smh oracle

54G //显示oracle文件夹的大小为54G

### export

#### SYNOPSIS

export [-fn] [name[=word]] ...

#### DESCRIPTION

设置或显示环境变量。该命令只对当前shell后的使用有效

#### OPTION

##### –f

变量名为函数名

##### –n

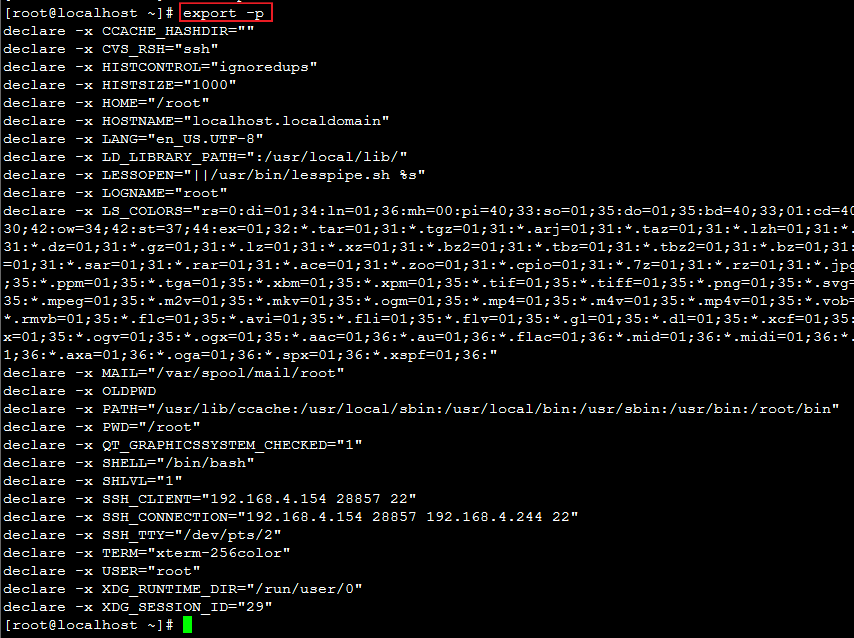
删除指定的变量。（没有永久删除，只是在export之后找不到该变量）

##### –p

列出所有shell赋予程序的环境变量。

#### EXAMPLES

##### 显示所有的环境变量



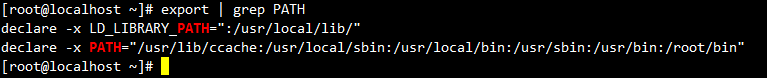
##### 修改环境变量

该命令只对当前shell后的使用有效

#export PATH=$PATH:your path

查看是否已经设好，可用命令export查看：

export | grep PATH



### find与xargs【未整理】

从根目录下开始查找abc.cpp文件，无错误输出

find / -name abc.cpp 2>/dev/null

在当前目录下所有.cpp文件中查找efg函数

find . -name "\*.cpp" | xargs grep 'efg'

xargs展开find获得的结果，使其作为grep的参数

另外 rm mv等命令对大量文件操作是报错 -bash: /bin/rm: Argument list too long

也可用xargs 解决

删除当前目录下所有.cpp文件

find . -name "\*.cpp" | xargs rm

在使用 find命令的-exec选项处理匹配到的文件时， find命令将所有匹配到的文件一起传递给exec执行。但有些系统对能够传递给exec的命令长度有限制，这样在find命令运行几分钟之后，就会出现溢出错误。错误信息通常是“参数列太长”或“参数列溢出”。这就是xargs命令的用处所在，特别是与find命令一起使用。

find命令把匹配到的文件传递给xargs命令，而xargs命令每次只获取一部分文件而不是全部，不像-exec选项那样。这样它可以先处理最先获取的一部分文件，然后是下一批，并如此继续下去。

在有些系统中，使用-exec选项会为处理每一个匹配到的文件而发起一个相应的进程，并非将匹配到的文件全部作为参数一次执行；这样在有些情况下就会出现进程过多，系统性能下降的问题，因而效率不高； 而使用xargs命令则只有一个进程。另外，在使用xargs命令时，究竟是一次获取所有的参数，还是分批取得参数，以及每一次获取参数的数目都会根据该命令的选项及系统内核中相应的可调参数来确定。

使用实例：

实例1： 查找系统中的每一个普通文件，然后使用xargs命令来测试它们分别属于哪类文件

命令：

find . -type f -print | xargs file

输出：

[root@localhost test]# ll

总计 312

-rw-r--r-- 1 root root 302108 11-03 06:19 log2012.log

-rw-r--r-- 1 root root 0 11-12 22:25 log2013.log

-rw-r--r-- 1 root root 0 11-12 22:25 log2014.log

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxrwx 2 root root 4096 11-12 19:32 test3

drwxrwxrwx 2 root root 4096 11-12 19:32 test4

[root@localhost test]# find . -type f -print | xargs file

./log2014.log: empty

./log2013.log: empty

./log2012.log: ASCII text

[root@localhost test]#

实例2：在整个系统中查找内存信息转储文件(core dump) ，然后把结果保存到/tmp/core.log 文件中

命令：

find / -name "core" -print | xargs echo "" >/tmp/core.log

输出：

[root@localhost test]# find / -name "core" -print | xargs echo "" >/tmp/core.log

[root@localhost test]# cd /tmp

[root@localhost tmp]# ll

总计 16

-rw-r--r-- 1 root root 1524 11-12 22:29 core.log

drwx------ 2 root root 4096 11-12 22:24 ssh-TzcZDx1766

drwx------ 2 root root 4096 11-12 22:28 ssh-ykiRPk1815

drwx------ 2 root root 4096 11-03 07:11 vmware-root

实例3:在当前目录下查找所有用户具有读、写和执行权限的文件，并收回相应的写权限

命令：

find . -perm -7 -print | xargs chmod o-w

输出：

[root@localhost test]# ll

总计 312

-rw-r--r-- 1 root root 302108 11-03 06:19 log2012.log

-rw-r--r-- 1 root root 0 11-12 22:25 log2013.log

-rw-r--r-- 1 root root 0 11-12 22:25 log2014.log

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxrwx 2 root root 4096 11-12 19:32 test3

drwxrwxrwx 2 root root 4096 11-12 19:32 test4

[root@localhost test]# find . -perm -7 -print | xargs chmod o-w

[root@localhost test]# ll

总计 312

-rw-r--r-- 1 root root 302108 11-03 06:19 log2012.log

-rw-r--r-- 1 root root 0 11-12 22:25 log2013.log

-rw-r--r-- 1 root root 0 11-12 22:25 log2014.log

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxr-x 2 root root 4096 11-12 19:32 test3

drwxrwxr-x 2 root root 4096 11-12 19:32 test4

[root@localhost test]#

说明：

执行命令后，文件夹scf、test3和test4的权限都发生改变

实例4：用grep命令在所有的普通文件中搜索hostname这个词

命令：

find . -type f -print | xargs grep "hostname"

输出：

[root@localhost test]# find . -type f -print | xargs grep "hostname"

./log2013.log:hostnamebaidu=baidu.com

./log2013.log:hostnamesina=sina.com

./log2013.log:hostnames=true[root@localhost test]#

实例5：用grep命令在当前目录下的所有普通文件中搜索hostnames这个词

命令：

find . -name \\* -type f -print | xargs grep "hostnames"

输出：

[root@peida test]# find . -name \\* -type f -print | xargs grep "hostnames"

./log2013.log:hostnamesina=sina.com

./log2013.log:hostnames=true[root@localhost test]#

说明：

注意，在上面的例子中， \用来取消find命令中的\*在shell中的特殊含义。

实例6：使用xargs执行mv

命令：

find . -name "\*.log" | xargs -i mv {} test4

输出：

[root@localhost test]# ll

总计 316

-rw-r--r-- 1 root root 302108 11-03 06:19 log2012.log

-rw-r--r-- 1 root root 61 11-12 22:44 log2013.log

-rw-r--r-- 1 root root 0 11-12 22:25 log2014.log

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxr-x 2 root root 4096 11-12 22:54 test3

drwxrwxr-x 2 root root 4096 11-12 19:32 test4

[root@localhost test]# cd test4/

[root@localhost test4]# ll

总计 0[root@localhost test4]# cd ..

[root@localhost test]# find . -name "\*.log" | xargs -i mv {} test4

[root@localhost test]# ll

总计 12drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxr-x 2 root root 4096 11-13 05:50 test3

drwxrwxr-x 2 root root 4096 11-13 05:50 test4

[root@localhost test]# cd test4/

[root@localhost test4]# ll

总计 304

-rw-r--r-- 1 root root 302108 11-12 22:54 log2012.log

-rw-r--r-- 1 root root 61 11-12 22:54 log2013.log

-rw-r--r-- 1 root root 0 11-12 22:54 log2014.log

[root@localhost test4]#

实例7：find后执行xargs提示xargs: argument line too long解决方法：

命令：

find . -type f -atime +0 -print0 | xargs -0 -l1 -t rm -f

输出：

[root@pd test4]# find . -type f -atime +0 -print0 | xargs -0 -l1 -t rm -f

rm -f

[root@pdtest4]#

说明：

-l1是一次处理一个；-t是处理之前打印出命令

实例8：使用-i参数默认的前面输出用{}代替，-I参数可以指定其他代替字符，如例子中的[]

命令：

输出：

[root@localhost test]# ll

总计 12drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxr-x 2 root root 4096 11-13 05:50 test3

drwxrwxr-x 2 root root 4096 11-13 05:50 test4

[root@localhost test]# cd test4

[root@localhost test4]# find . -name "file" | xargs -I [] cp [] ..

[root@localhost test4]# ll

总计 304

-rw-r--r-- 1 root root 302108 11-12 22:54 log2012.log

-rw-r--r-- 1 root root 61 11-12 22:54 log2013.log

-rw-r--r-- 1 root root 0 11-12 22:54 log2014.log

[root@localhost test4]# cd ..

[root@localhost test]# ll

总计 316

-rw-r--r-- 1 root root 302108 11-13 06:03 log2012.log

-rw-r--r-- 1 root root 61 11-13 06:03 log2013.log

-rw-r--r-- 1 root root 0 11-13 06:03 log2014.log

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxr-x 2 root root 4096 11-13 05:50 test3

drwxrwxr-x 2 root root 4096 11-13 05:50 test4

[root@localhost test]#

说明：

使用-i参数默认的前面输出用{}代替，-I参数可以指定其他代替字符，如例子中的[]

实例9：xargs的-p参数的使用

命令：

find . -name "\*.log" | xargs -p -i mv {} ..

输出：

[root@localhost test3]# ll

总计 0

-rw-r--r-- 1 root root 0 11-13 06:06 log2015.log

[root@localhost test3]# cd ..

[root@localhost test]# ll

总计 316

-rw-r--r-- 1 root root 302108 11-13 06:03 log2012.log

-rw-r--r-- 1 root root 61 11-13 06:03 log2013.log

-rw-r--r-- 1 root root 0 11-13 06:03 log2014.log

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxr-x 2 root root 4096 11-13 06:06 test3

drwxrwxr-x 2 root root 4096 11-13 05:50 test4

[root@localhost test]# cd test3

[root@localhost test3]# find . -name "\*.log" | xargs -p -i mv {} ..

mv ./log2015.log .. ?...y

[root@localhost test3]# ll

总计 0[root@localhost test3]# cd ..

[root@localhost test]# ll

总计 316

-rw-r--r-- 1 root root 302108 11-13 06:03 log2012.log

-rw-r--r-- 1 root root 61 11-13 06:03 log2013.log

-rw-r--r-- 1 root root 0 11-13 06:03 log2014.log

-rw-r--r-- 1 root root 0 11-13 06:06 log2015.log

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxr-x 2 root root 4096 11-13 06:08 test3

drwxrwxr-x 2 root root 4096 11-13 05:50 test4

[root@localhost test]#

说明：

-p参数会提示让你确认是否执行后面的命令,y执行，n不执行。

### grep

#### 作用

Linux系统中grep命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。grep全称是Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

#### 格式

grep [options]

#### 主要参数

##### option参数

－c：只输出匹配行的计数。

－I：不区分大 小写(只适用于单字符)。

－h：查询多文件时不显示文件名。

－l：查询多文件时只输出包含匹配字符的文件名。

－n：显示匹配行及 行号。

－s：不显示不存在或无匹配文本的错误信息。

－v：显示不包含匹配文本的所有行。

##### pattern正则表达式主要参数

\： 忽略正则表达式中特殊字符的原有含义。

^：匹配正则表达式的开始行。

$: 匹配正则表达式的结束行。

\<：从匹配正则表达 式的行开始。

\>：到匹配正则表达式的行结束。

[ ]：单个字符，如[A]即A符合要求 。

[ - ]：范围，如[A-Z]，即A、B、C一直到Z都符合要求 。

。：所有的单个字符。

\* ：有字符，长度可以为0。

#### 简单实例

##### grep ‘test’ d\*

显示所有以d开头的文件中包含 test的行。

##### grep ‘test’ aa bb cc

显示在aa，bb，cc文件中匹配test的行。

##### grep ‘[a-z]\{5\}’ aa

显示所有包含每个字符串至少有5个连续小写字符的字符串的行。

##### grep ‘w\(es\)t.\*\1′ aa

如果west被匹配，则es就被存储到内存中，并标记为1，然后搜索任意个字符(.\*)，这些字符后面紧跟着 另外一个es(\1)，找到就显示该行。如果用egrep或grep -E，就不用”\”号进行转义，直接写成’w(es)t.\*\1′就可以了。

#### 复杂实例

假设您正在’/usr/src/Linux/Doc’目录下搜索带字符 串’magic’的文件：

##### $ grep magic /usr/src/Linux/Doc/\*

sysrq.txt:\* How do I enable the magic SysRQ key?

sysrq.txt:\* How do I use the magic SysRQ key?

其中文件’sysrp.txt’包含该字符串，讨论的是 SysRQ 的功能。

默认情况下，’grep’只搜索当前目录。如果 此目录下有许多子目录，’grep’会以如下形式列出：

grep: sound: Is a directory

这可能会使’grep’ 的输出难于阅读。这里有两种解决的办法：

明确要求搜索子目录：grep -r

或忽略子目录：grep -d skip

如果有很多 输出时，您可以通过管道将其转到’less’上阅读：

##### $ grep magic /usr/src/Linux/Documentation/\* | less

这样，您就可以更方便地阅读。

有一点要注意，您必需提供一个文件过滤方式(搜索全部文件的话用 \*)。如果您忘了，’grep’会一直等着，直到该程序被中断。如果您遇到了这样的情况，按 <CTRL c> ，然后再试。

下面还有一些有意思的命令行参数：

##### grep -i pattern files ：

不区分大小写地搜索。默认情况区分大小写，

##### grep -l pattern files ：

只列出匹配的文件名，

##### grep -L pattern files ：

列出不匹配的文件名，

##### grep -w pattern files ：

只匹配整个单词，而不是字符串的一部分(如匹配’magic’，而不是’magical’)，

##### grep -C number pattern files ：

匹配的上下文分别显示[number]行，

##### grep pattern1 | pattern2 files ：

显示匹配 pattern1 或 pattern2 的行，

##### grep pattern1 files | grep pattern2：

显示既匹配 pattern1 又匹配 pattern2 的行。

##### grep -n pattern files

即可显示行号信息

##### grep -c pattern files

即可查找总行数

这里还有些用于搜索的特殊符号：

\< 和 \> 分别标注单词的开始与结尾。

例如：

grep man \* 会匹配 ‘Batman’、’manic’、’man’等，

grep ‘\<man’ \* 匹配’manic’和’man’，但不是’Batman’，

grep ‘\<man\>’ 只匹配’man’，而不是’Batman’或’manic’等其他的字符串。

‘^’：指匹配的字符串在行首，

‘$’：指匹配的字符串在行 尾，

### less

#### SYNOPSIS

ls -al /etc | less

#### DESCRIPTION

less 是一个分页工具，它允许你一页一页（或一个屏幕一个屏幕）地查看信息。

* 要**向下移动一个屏幕**，按 [**Space**] 键；
* 要**向上移动一个屏幕**，按 [**b**] 键；
* 要**向下移动一行**，按 [**n**] 键；
* 要**移动到最开始**，按 [**p**] 键；
* 要**移动到最尾**，按 [**G**] 键；
* 要**退出**，按 [**q**] 键。
* 使用 less 命令时，你还可以使用**箭头键**来前后移动。
* 要使用 less 来搜索文本文件的输出，按 [**/**]，然后键入你想在文件内搜索的内容。譬如：/Linux

### ls

#### SYNOPSIS

ls [ OPTION ] ... [ FILE ] ...

#### DESCRIPTION

#### OPTION

**-l（long）**

长格式，显示**文件类型**、**权限**、硬链接的数目、文件拥有者、**文件所在的组、大小**、**日期**和**文件名**。

**brw-r--r-- 1 unixguy staff 64, 64 Jan 27 05:52 block**

**crw-r--r-- 1 unixguy staff 64, 255 Jan 26 13:57 character**

**-rw-r--r-- 1 unixguy staff 290 Jan 26 14:08 compressed.gz**

**-rw-r--r-- 1 unixguy staff 331836 Jan 26 14:06 data.ppm**

**drwxrwx--x 2 unixguy staff 48 Jan 26 11:28 directory**

**-rwxrwx--x 1 unixguy staff 29 Jan 26 14:03 executable**

**prw-r--r-- 1 unixguy staff 0 Jan 26 11:50 fifo**

**lrwxrwxrwx 1 unixguy staff 3 Jan 26 11:44 link -> dir**

**-rw-rw---- 1 unixguy staff 217 Jan 26 14:08 regularfile**

**-F**

在不同类型的文件的文件名结尾追加一个字符以示区别。可执行文件后加"\*"，目录后加"/"，管道文件后加"|"，套接字文件后加"="，普通文件没有后缀。

**-a（all）**

显示所有文件，包括以 . 开头的文件名（默认不显示）。

**-R（recursive）**

迭代显示目录下所有的子目录。ls -R／会显示文件系统中的所有文件。

**-d（directory）**

显示目录本身的信息，而不是列出目录下的文件。

#### EXAMPLES

#### 常用

1. 显示彩色目录列表

打开/etc/bashrc, 加入如下一行:

alias ls="ls --color"

下次启动bash时就可以像在Slackware里那样显示彩色的目录列表了, 其中颜色的含义如下:

1. 蓝色-->目录

2. 绿色-->可执行文件

3. 红色-->压缩文件

4. 浅蓝色-->链接文件

5. 灰色-->其他文件

### make

#### SYNOPSIS

make [ -f makefile ] [ options ] ... [ targets ] ...

#### DESCRIPTION

解压缩文件

GNU `tar' saves many files together into a single tape or disk archive, and can restore individual files from the archive.

#### OPTION

##### -f file

--file=file, --makefile=FILE

Use file as a makefile.

##### -B

先clean，再make

#### EXAMPLES

##### 编译文件

make

##### 编译指定文件

make –f Makefile\_test

### more

#### SYNOPSIS

ls -al /etc | more

#### DESCRIPTION

more 和 less 的主要区别是，less 允许你使用箭头键来前后移动，而 more 使用[Spacebar]和[b]键来前后移动。

使用 ls 和 more 命令来列举 /etc 目录中的内容。

ls -al /etc | more

要使用 more 在文本文件内搜索关键字，按 [/] 并键入搜索条目。譬如：

/foo

使用 [Spacebar] 来向前翻阅页码。按 [q] 来退出。

#### EXAMPLES

### rpm

#### SYNOPSIS

##### QUERYING AND VERIFYING PACKAGES:

rpm {-q|--query} [select-options] [query-options]

rpm {-V|--verify} [select-options] [verify-options]

rpm [--nosignature] [--nodigest]

PACKAGE\_FILE ...

##### INSTALLING, UPGRADING, AND REMOVING PACKAGES:

rpm {-i|--install} [install-options] PACKAGE\_FILE ...

rpm {-U|--upgrade} [install-options] PACKAGE\_FILE ...

rpm {-F|--freshen} [install-options] PACKAGE\_FILE ...

rpm {-e|--erase} [--allmatches] [--nodeps] [--noscripts]

[--notriggers] [--test] PACKAGE\_NAME ...

##### MISCELLANEOUS:

rpm {--querytags|--showrc}

rpm {--setperms|--setugids} PACKAGE\_NAME ...

##### select-options

[PACKAGE\_NAME] [-a,--all] [-f,--file FILE]

[-g,--group GROUP] {-p,--package PACKAGE\_FILE]

[--hdrid SHA1] [--pkgid MD5] [--tid TID]

[--querybynumber HDRNUM] [--triggeredby PACKAGE\_NAME]

[--whatprovides CAPABILITY] [--whatrequires CAPABILITY]

##### query-options

rpm {--setperms|--setugids} PACKAGE\_NAME ...

##### select-options

[PACKAGE\_NAME] [-a,--all] [-f,--file FILE]

[-g,--group GROUP] {-p,--package PACKAGE\_FILE]

[--hdrid SHA1] [--pkgid MD5] [--tid TID]

[--querybynumber HDRNUM] [--triggeredby PACKAGE\_NAME]

[--whatprovides CAPABILITY] [--whatrequires CAPABILITY]

##### query-options

[--changelog] [-c,--configfiles] [--conflicts]

[-d,--docfiles] [--dump] [--filesbypkg] [-i,--info]

[--last] [-l,--list] [--obsoletes] [--provides]

[--qf,--queryformat QUERYFMT] [-R,--requires]

[--scripts] [-s,--state] [--triggers,--triggerscripts]

##### verify-options

[--nodeps] [--nofiles] [--noscripts]

[--nodigest] [--nosignature]

[--nolinkto] [--nofiledigest] [--nosize] [--nouser]

[--nogroup] [--nomtime] [--nomode] [--nordev]

[--nocaps]

##### install-options

[--allfiles] [--badreloc] [--excludepath OLDPATH]

[--excludedocs] [--force] [-h,--hash]

[--ignoresize] [--ignorearch] [--ignoreos]

[--includedocs] [--justdb] [--nocollections]

[--nodeps] [--nodigest] [--nosignature]

[--noorder] [--noscripts] [--notriggers]

[--oldpackage] [--percent] [--prefix NEWPATH]

[--relocate OLDPATH=NEWPATH]

[--replacefiles] [--replacepkgs]

[--test]

#### DESCRIPTION

rpm is a powerful Package Manager, which can be used to build, install, query, verify, update, and erase individual software packages.

#### OPTION

##### –c

--create

create a new archive

建立一个压缩文件的参数指令

注意：不能与x/t共存

##### –x

--extract

extract files from an archive

解开一个压缩文件的参数指令

注意：不能与c/t共存

#### EXAMPLES

##### 安装一个包

rpm –ivh \*.rpm

##### 升级一个包

rpm –Uvh \*.rpm

##### 卸载一个包

rpm –e \*

##### 安装参数

--force 即使覆盖属于其它包的文件也强迫安装

--nodeps 如果该RPM包的安装依赖其它包，即使其它包没装，也强迫安装。

##### 查询一个包是否被安装

rpm -q < rpm package name>

##### 得到被安装的包的信息

rpm -qi < rpm package name>

##### 列出该包中有哪些文件

rpm -ql < rpm package name>

##### 列出服务器上的一个文件属于哪一个RPM包

rpm -qf

##### 可综合好几个参数一起用

rpm -qil < rpm package name>

##### 列出所有被安装的rpm package

rpm -qa

##### 列出一个未被安装进系统的RPM包文件中包含有哪些文件？

rpm -qilp < rpm package name>

### tar

#### SYNOPSIS

tar [option] [file]

#### DESCRIPTION

解压缩文件

GNU `tar' saves many files together into a single tape or disk archive, and can restore individual files from the archive.

#### OPTION

##### –c

--create

create a new archive

建立一个压缩文件的参数指令

注意：不能与x/t共存

##### –x

--extract

extract files from an archive

解开一个压缩文件的参数指令

注意：不能与c/t共存

##### –t

--list

list the contents of an archive

显示文件列表

注意：不能与c/x共存

##### –f

--file=ARCHIVE

use archive file or device ARCHIVE

OPTION要以f结尾，目前我见到的都是这样

##### –v

--verbose

verbosely list files processed

显示文件的权限，大小，创建时间

##### –j

--bzip2

filter the archive through bzip2

处理属性为bzip2的文件，即常见的\*.tar.bz2文件

##### –z

--gzip

filter the archive through gzip

处理属性为gzip的文件，即常见的\*.tar.gz文件

##### -N

--newer=DATE-OR-FILE, --after-date=DATE-OR-FILE

only store files newer than DATE-OR-FILE

比后面接的日期(yyyy/mm/dd)还要新的才会被打包进新建的文件中

#### EXAMPLES

##### 解压以tar.gz结尾的文件

tar -x**z**vf \*.tar.gz

##### 解压以tar.bz2结尾的文件

tar -x**j**vf \*.tar.bz2

##### Create archive.tar from files foo and bar.

tar -cf archive.tar foo bar

##### List all files in archive.tar verbosely.

tar -tvf archive.tar

##### 将整个 /etc 目录下的文件全部打包成为/tmp/etc.tar

tar -cf /tmp/etc.tar /etc

### tcpdump

#### SYNOPSIS

tar [option] [file]

#### DESCRIPTION

用简单的话来定义tcpdump，就是：dump the traffic on a network，根据使用者的定义对网络上的数据包进行截获的包分析工具。 tcpdump可以将网络中传送的数据包的“头”完全截获下来提供分析。它支持针对网络层、协议、主机、网络或端口的过滤，并提供and、or、not等逻辑语句来帮助你去掉无用的信息。

#### OPTION

##### –c

--create

create a new archive

建立一个压缩文件的参数指令

注意：不能与x/t共存

##### –x

--extract

extract files from an archive

解开一个压缩文件的参数指令

注意：不能与c/t共存

#### EXAMPLES

##### **默认启动**

tcpdump

普通情况下，直接启动tcpdump将监视第一个网络接口上所有流过的数据包。

##### **监视指定网络接口的数据包**

tcpdump -i eth1

如果不指定网卡，默认tcpdump只会监视第一个网络接口，一般是eth0，下面的例子都没有指定网络接口。

##### **监视指定主机的数据包**

打印所有进入或离开sundown的数据包.

tcpdump host sundown

也可以指定ip,例如截获所有210.27.48.1 的主机收到的和发出的所有的数据包

tcpdump host 210.27.48.1

打印helios 与 hot 或者与 ace 之间通信的数据包

tcpdump host helios and \( hot or ace \)

截获主机210.27.48.1 和主机210.27.48.2 或210.27.48.3的通信

tcpdump host 210.27.48.1 and \ (210.27.48.2 or 210.27.48.3 \)

打印ace与任何其他主机之间通信的IP 数据包, 但不包括与helios之间的数据包.

tcpdump ip host ace and not helios

如果想要获取主机210.27.48.1除了和主机210.27.48.2之外所有主机通信的ip包，使用命令：

tcpdump ip host 210.27.48.1 and ! 210.27.48.2

截获主机hostname发送的所有数据

tcpdump -i eth0 src host hostname

监视所有送到主机hostname的数据包

tcpdump -i eth0 dst host hostname

##### **监视指定主机和端口的数据包**

如果想要获取主机210.27.48.1接收或发出的telnet包，使用如下命令

tcpdump tcp port 23 host 210.27.48.1

对本机的udp 123 端口进行监视 123 为ntp的服务端口

tcpdump udp port 123

##### **监视指定网络的数据包**

打印本地主机与Berkeley网络上的主机之间的所有通信数据包(nt: ucb-ether, 此处可理解为'Berkeley网络'的网络地址,此表达式最原始的含义可表达为: 打印网络地址为ucb-ether的所有数据包)

tcpdump net ucb-ether

打印所有通过网关snup的ftp数据包(注意, 表达式被单引号括起来了, 这可以防止shell对其中的括号进行错误解析)

tcpdump 'gateway snup and (port ftp or ftp-data)'

打印所有源地址或目标地址是本地主机的IP数据包

(如果本地网络通过网关连到了另一网络, 则另一网络并不能算作本地网络.(nt: 此句翻译曲折,需补充).localnet 实际使用时要真正替换成本地网络的名字)

tcpdump ip and not net localnet

##### **监视指定协议的数据包**

打印TCP会话中的的开始和结束数据包, 并且数据包的源或目的不是本地网络上的主机.(nt: localnet, 实际使用时要真正替换成本地网络的名字))

tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net localnet'

打印所有源或目的端口是80, 网络层协议为IPv4, 并且含有数据,而不是SYN,FIN以及ACK-only等不含数据的数据包.(ipv6的版本的表达式可做练习)

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

(nt: 可理解为, ip[2:2]表示整个ip数据包的长度, (ip[0]&0xf)<<2)表示ip数据包包头的长度(ip[0]&0xf代表包中的IHL域, 而此域的单位为32bit, 要换算

成字节数需要乘以4,　即左移2.　(tcp[12]&0xf0)>>4 表示tcp头的长度, 此域的单位也是32bit,　换算成比特数为 ((tcp[12]&0xf0) >> 4)　<<　２,

即 ((tcp[12]&0xf0)>>2).　((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0　表示: 整个ip数据包的长度减去ip头的长度,再减去

tcp头的长度不为0, 这就意味着, ip数据包中确实是有数据.对于ipv6版本只需考虑ipv6头中的'Payload Length' 与 'tcp头的长度'的差值, 并且其中表达方式'ip[]'需换成'ip6[]'.)

打印长度超过576字节, 并且网关地址是snup的IP数据包

tcpdump 'gateway snup and ip[2:2] > 576'

打印所有IP层广播或多播的数据包， 但不是物理以太网层的广播或多播数据报

tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'

打印除'echo request'或者'echo reply'类型以外的ICMP数据包( 比如,需要打印所有非ping 程序产生的数据包时可用到此表达式 .

(nt: 'echo reuqest' 与 'echo reply' 这两种类型的ICMP数据包通常由ping程序产生))

tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'

##### **tcpdump 与wireshark**

Wireshark(以前是ethereal)是Windows下非常简单易用的抓包工具。但在Linux下很难找到一个好用的图形化抓包工具。

还好有Tcpdump。我们可以用Tcpdump + Wireshark 的完美组合实现：在 Linux 里抓包，然后在Windows 里分析包。

tcpdump tcp -i eth1 -t -s 0 -c 100 and dst port ! 22 and src net 192.168.1.0/24 -w ./target.cap

(1)tcp: ip icmp arp rarp 和 tcp、udp、icmp这些选项等都要放到第一个参数的位置，用来过滤数据报的类型

(2)-i eth1 : 只抓经过接口eth1的包

(3)-t : 不显示时间戳

(4)-s 0 : 抓取数据包时默认抓取长度为68字节。加上-S 0 后可以抓到完整的数据包

(5)-c 100 : 只抓取100个数据包

(6)dst port ! 22 : 不抓取目标端口是22的数据包

(7)src net 192.168.1.0/24 : 数据包的源网络地址为192.168.1.0/24

(8)-w ./target.cap : 保存成cap文件，方便用ethereal(即wireshark)分析

##### **使用tcpdump抓取HTTP包**

tcpdump -XvvennSs 0 -i eth0 tcp[20:2]=0x4745 or tcp[20:2]=0x4854

0x4745 为"GET"前两个字母"GE",0x4854 为"HTTP"前两个字母"HT"。

tcpdump 对截获的数据并没有进行彻底解码，数据包内的大部分内容是使用十六进制的形式直接打印输出的。显然这不利于分析网络故障，通常的解决办法是先使用带-w参数的tcpdump 截获数据并保存到文件中，然后再使用其他程序(如Wireshark)进行解码分析。当然也应该定义过滤规则，以避免捕获的数据包填满整个硬盘。

#### 输出信息含义

首先我们注意一下，基本上tcpdump总的的输出格式为：系统时间 来源主机.端口 > 目标主机.端口 数据包参数

tcpdump 的输出格式与协议有关.以下简要描述了大部分常用的格式及相关例子.

##### ****链路层头****

对于FDDI网络, '-e' 使tcpdump打印出指定数据包的'frame control' 域, 源和目的地址, 以及包的长度.(frame control域

控制对包中其他域的解析). 一般的包(比如那些IP datagrams)都是带有'async'(异步标志)的数据包，并且有取值0到7的优先级;

比如 'async4'就代表此包为异步数据包，并且优先级别为4. 通常认为,这些包们会内含一个 LLC包(逻辑链路控制包); 这时,如果此包

不是一个ISO datagram或所谓的SNAP包，其LLC头部将会被打印(nt:应该是指此包内含的 LLC包的包头).

对于Token Ring网络(令牌环网络), '-e' 使tcpdump打印出指定数据包的'frame control'和'access control'域, 以及源和目的地址,

外加包的长度. 与FDDI网络类似, 此数据包通常内含LLC数据包. 不管 是否有'-e'选项.对于此网络上的'source-routed'类型数据包(nt:

意译为:源地址被追踪的数据包,具体含义未知,需补充), 其包的源路由信息总会被打印.

对于802.11网络(WLAN,即wireless local area network), '-e' 使tcpdump打印出指定数据包的'frame control域,

包头中包含的所有地址, 以及包的长度.与FDDI网络类似, 此数据包通常内含LLC数据包.

(注意: 以下的描述会假设你熟悉SLIP压缩算法 (nt:SLIP为Serial Line Internet Protocol.), 这个算法可以在

RFC-1144中找到相关的蛛丝马迹.)

对于SLIP网络(nt:SLIP links, 可理解为一个网络, 即通过串行线路建立的连接, 而一个简单的连接也可看成一个网络),

数据包的'direction indicator'('方向指示标志')("I"表示入, "O"表示出), 类型以及压缩信息将会被打印. 包类型会被首先打印.

类型分为ip, utcp以及ctcp(nt:未知, 需补充). 对于ip包,连接信息将不被打印(nt:SLIP连接上,ip包的连接信息可能无用或没有定义.

reconfirm).对于TCP数据包, 连接标识紧接着类型表示被打印. 如果此包被压缩, 其被编码过的头部将被打印.

此时对于特殊的压缩包,会如下显示:

\*S+n 或者 \*SA+n, 其中n代表包的(顺序号或(顺序号和应答号))增加或减少的数目(nt | rt:S,SA拗口, 需再译).

对于非特殊的压缩包,0个或更多的'改变'将会被打印.'改变'被打印时格式如下:

'标志'+/-/=n 包数据的长度 压缩的头部长度.

其中'标志'可以取以下值:

U(代表紧急指针), W(指缓冲窗口), A(应答), S(序列号), I(包ID),而增量表达'=n'表示被赋予新的值, +/-表示增加或减少.

比如, 以下显示了对一个外发压缩TCP数据包的打印, 这个数据包隐含一个连接标识(connection identifier); 应答号增加了6,

顺序号增加了49, 包ID号增加了6; 包数据长度为3字节(octect), 压缩头部为6字节.(nt:如此看来这应该不是一个特殊的压缩数据包).

ARP/RARP 数据包

tcpdump对Arp/rarp包的输出信息中会包含请求类型及该请求对应的参数. 显示格式简洁明了. 以下是从主机rtsg到主机csam的'rlogin'

(远程登录)过程开始阶段的数据包样例:

arp who-has csam tell rtsg

arp reply csam is-at CSAM

第一行表示:rtsg发送了一个arp数据包(nt:向全网段发送,arp数据包）以询问csam的以太网地址

Csam（nt:可从下文看出来, 是Csam）以她自己的以太网地址做了回应(在这个例子中, 以太网地址以大写的名字标识, 而internet

地址(即ip地址)以全部的小写名字标识).

如果使用tcpdump -n, 可以清晰看到以太网以及ip地址而不是名字标识:

arp who-has 128.3.254.6 tell 128.3.254.68

arp reply 128.3.254.6 is-at 02:07:01:00:01:c4

如果我们使用tcpdump -e, 则可以清晰的看到第一个数据包是全网广播的, 而第二个数据包是点对点的:

RTSG Broadcast 0806 64: arp who-has csam tell rtsg

CSAM RTSG 0806 64: arp reply csam is-at CSAM

第一个数据包表明:以arp包的源以太地址是RTSG, 目标地址是全以太网段, type域的值为16进制0806(表示ETHER\_ARP(nt:arp包的类型标识)),

包的总长度为64字节.

##### ****TCP 数据包****

(注意:以下将会假定你对 RFC-793所描述的TCP熟悉. 如果不熟, 以下描述以及tcpdump程序可能对你帮助不大.(nt:警告可忽略,

只需继续看, 不熟悉的地方可回头再看.).

通常tcpdump对tcp数据包的显示格式如下:

src > dst: flags data-seqno ack window urgent options

src 和 dst 是源和目的IP地址以及相应的端口. flags 标志由S(SYN), F(FIN), P(PUSH, R(RST),

W(ECN CWT(nt | rep:未知, 需补充))或者 E(ECN-Echo(nt | rep:未知,　需补充))组成,

单独一个'.'表示没有flags标识. 数据段顺序号(Data-seqno)描述了此包中数据所对应序列号空间中的一个位置(nt:整个数据被分段,

每段有一个顺序号, 所有的顺序号构成一个序列号空间)(可参考以下例子). Ack 描述的是同一个连接,同一个方向,下一个本端应该接收的

(对方应该发送的)数据片段的顺序号. Window是本端可用的数据接收缓冲区的大小(也是对方发送数据时需根据这个大小来组织数据).

Urg(urgent) 表示数据包中有紧急的数据. options 描述了tcp的一些选项, 这些选项都用尖括号来表示(如 <mss 1024>).

src, dst 和 flags 这三个域总是会被显示. 其他域的显示与否依赖于tcp协议头里的信息.

这是一个从trsg到csam的一个rlogin应用登录的开始阶段.

rtsg.1023 > csam.login: S 768512:768512(0) win 4096 <mss 1024>

csam.login > rtsg.1023: S 947648:947648(0) ack 768513 win 4096 <mss 1024>

rtsg.1023 > csam.login: . ack 1 win 4096

rtsg.1023 > csam.login: P 1:2(1) ack 1 win 4096

csam.login > rtsg.1023: . ack 2 win 4096

rtsg.1023 > csam.login: P 2:21(19) ack 1 win 4096

csam.login > rtsg.1023: P 1:2(1) ack 21 win 4077

csam.login > rtsg.1023: P 2:3(1) ack 21 win 4077 urg 1

csam.login > rtsg.1023: P 3:4(1) ack 21 win 4077 urg 1

第一行表示有一个数据包从rtsg主机的tcp端口1023发送到了csam主机的tcp端口login上(nt:udp协议的端口和tcp协议的端

口是分别的两个空间, 虽然取值范围一致). S表示设置了SYN标志. 包的顺序号是768512, 并且没有包含数据.(表示格式

为:'first:last(nbytes)', 其含义是'此包中数据的顺序号从first开始直到last结束，不包括last. 并且总共包含nbytes的

用户数据'.) 没有捎带应答(nt:从下文来看，第二行才是有捎带应答的数据包), 可用的接受窗口的大小为4096bytes, 并且请求端(rtsg)

的最大可接受的数据段大小是1024字节(nt:这个信息作为请求发向应答端csam, 以便双方进一步的协商).

Csam 向rtsg 回复了基本相同的SYN数据包, 其区别只是多了一个' piggy-backed ack'(nt:捎带回的ack应答, 针对rtsg的SYN数据包).

rtsg 同样针对csam的SYN数据包回复了一ACK数据包作为应答. '.'的含义就是此包中没有标志被设置. 由于此应答包中不含有数据, 所以

包中也没有数据段序列号. 提醒! 此ACK数据包的顺序号只是一个小整数1. 有如下解释:tcpdump对于一个tcp连接上的会话, 只打印会话两端的

初始数据包的序列号,其后相应数据包只打印出与初始包序列号的差异.即初始序列号之后的序列号,　可被看作此会话上当前所传数据片段在整个

要传输的数据中的'相对字节'位置(nt:双方的第一个位置都是1, 即'相对字节'的开始编号).　'-Ｓ'将覆盖这个功能,

使数据包的原始顺序号被打印出来.

第六行的含义为:rtsg 向 csam发送了19字节的数据(字节的编号为2到20，传送方向为rtsg到csam). 包中设置了PUSH标志. 在第7行,

csam 喊到， 她已经从rtsg中收到了21以下的字节, 但不包括21编号的字节. 这些字节存放在csam的socket的接收缓冲中, 相应地,

csam的接收缓冲窗口大小会减少19字节(nt:可以从第5行和第7行win属性值的变化看出来). csam在第7行这个包中也向rtsg发送了一个

字节. 在第8行和第9行, csam 继续向rtsg 分别发送了两个只包含一个字节的数据包, 并且这个数据包带PUSH标志.

如果所抓到的tcp包(nt:即这里的snapshot)太小了，以至tcpdump无法完整得到其头部数据, 这时, tcpdump会尽量解析这个不完整的头,

并把剩下不能解析的部分显示为'[|tcp]'. 如果头部含有虚假的属性信息(比如其长度属性其实比头部实际长度长或短), tcpdump会为该头部

显示'[bad opt]'. 如果头部的长度告诉我们某些选项(nt | rt:从下文来看， 指tcp包的头部中针对ip包的一些选项, 回头再翻)会在此包中,

而真正的IP(数据包的长度又不够容纳这些选项, tcpdump会显示'[bad hdr length]'.

抓取带有特殊标志的的TCP包(如SYN-ACK标志, URG-ACK标志等).

在TCP的头部中, 有8比特(bit)用作控制位区域, 其取值为:

CWR | ECE | URG | ACK | PSH | RST | SYN | FIN

(nt | rt:从表达方式上可推断:这8个位是用或的方式来组合的, 可回头再翻)

现假设我们想要监控建立一个TCP连接整个过程中所产生的数据包. 可回忆如下:TCP使用3次握手协议来建立一个新的连接; 其与此三次握手

连接顺序对应，并带有相应TCP控制标志的数据包如下:

1) 连接发起方(nt:Caller)发送SYN标志的数据包

2) 接收方(nt:Recipient)用带有SYN和ACK标志的数据包进行回应

3) 发起方收到接收方回应后再发送带有ACK标志的数据包进行回应

0 15 31

-----------------------------------------------------------------

| source port | destination port |

-----------------------------------------------------------------

| sequence number |

-----------------------------------------------------------------

| acknowledgment number |

-----------------------------------------------------------------

| HL | rsvd |C|E|U|A|P|R|S|F| window size |

-----------------------------------------------------------------

| TCP checksum | urgent pointer |

-----------------------------------------------------------------

一个TCP头部,在不包含选项数据的情况下通常占用20个字节(nt | rt:options 理解为选项数据，需回译). 第一行包含0到3编号的字节,

第二行包含编号4-7的字节.

如果编号从0开始算, TCP控制标志位于13字节(nt:第四行左半部分).

0 7| 15| 23| 31

----------------|---------------|---------------|----------------

| HL | rsvd |C|E|U|A|P|R|S|F| window size |

----------------|---------------|---------------|----------------

| | 13th octet | | |

让我们仔细看看编号13的字节:

| |

|---------------|

|C|E|U|A|P|R|S|F|

|---------------|

|7 5 3 0|

这里有我们感兴趣的控制标志位. 从右往左这些位被依次编号为0到7, 从而 PSH位在3号, 而URG位在5号.

提醒一下自己, 我们只是要得到包含SYN标志的数据包. 让我们看看在一个包的包头中, 如果SYN位被设置, 到底

在13号字节发生了什么:

|C|E|U|A|P|R|S|F|

|---------------|

|0 0 0 0 0 0 1 0|

|---------------|

|7 6 5 4 3 2 1 0|

在控制段的数据中, 只有比特1(bit number 1)被置位.

假设编号为13的字节是一个8位的无符号字符型,并且按照网络字节号排序(nt:对于一个字节来说，网络字节序等同于主机字节序), 其二进制值

如下所示:

00000010

并且其10进制值为:

0\*2^7 + 0\*2^6 + 0\*2^5 + 0\*2^4 + 0\*2^3 + 0\*2^2 + 1\*2^1 + 0\*2^0 = 2(nt: 1 \* 2^6 表示1乘以2的6次方, 也许这样更

清楚些, 即把原来表达中的指数7 6 ... 0挪到了下面来表达)

接近目标了, 因为我们已经知道, 如果数据包头部中的SYN被置位, 那么头部中的第13个字节的值为2(nt: 按照网络序, 即大头方式, 最重要的字节

在前面(在前面,即该字节实际内存地址比较小, 最重要的字节,指数学表示中数的高位, 如356中的3) ).

表达为tcpdump能理解的关系式就是:

tcp[13] 2

从而我们可以把此关系式当作tcpdump的过滤条件, 目标就是监控只含有SYN标志的数据包:

tcpdump -i xl0 tcp[13] 2 (nt: xl0 指网络接口, 如eth0)

这个表达式是说"让TCP数据包的第13个字节拥有值2吧", 这也是我们想要的结果.

现在, 假设我们需要抓取带SYN标志的数据包, 而忽略它是否包含其他标志.(nt:只要带SYN就是我们想要的). 让我们来看看当一个含有

SYN-ACK的数据包(nt:SYN 和 ACK 标志都有), 来到时发生了什么:

|C|E|U|A|P|R|S|F|

|---------------|

|0 0 0 1 0 0 1 0|

|---------------|

|7 6 5 4 3 2 1 0|

13号字节的1号和4号位被置位, 其二进制的值为:

00010010

转换成十进制就是:

0\*2^7 + 0\*2^6 + 0\*2^5 + 1\*2^4 + 0\*2^3 + 0\*2^2 + 1\*2^1 + 0\*2 = 18(nt: 1 \* 2^6 表示1乘以2的6次方, 也许这样更

清楚些, 即把原来表达中的指数7 6 ... 0挪到了下面来表达)

现在, 却不能只用'tcp[13] 18'作为tcpdump的过滤表达式, 因为这将导致只选择含有SYN-ACK标志的数据包, 其他的都被丢弃.

提醒一下自己, 我们的目标是: 只要包的SYN标志被设置就行, 其他的标志我们不理会.

为了达到我们的目标, 我们需要把13号字节的二进制值与其他的一个数做AND操作(nt:逻辑与)来得到SYN比特位的值. 目标是:只要SYN 被设置

就行, 于是我们就把她与上13号字节的SYN值(nt: 00000010).

00010010 SYN-ACK 00000010 SYN

AND 00000010 (we want SYN) AND 00000010 (we want SYN)

-------- --------

= 00000010 = 00000010

我们可以发现, 不管包的ACK或其他标志是否被设置, 以上的AND操作都会给我们相同的值, 其10进制表达就是2(2进制表达就是00000010).

从而我们知道, 对于带有SYN标志的数据包, 以下的表达式的结果总是真(true):

( ( value of octet 13 ) AND ( 2 ) ) ( 2 ) (nt: value of octet 13, 即13号字节的值)

灵感随之而来, 我们于是得到了如下的tcpdump 的过滤表达式

tcpdump -i xl0 'tcp[13] & 2 2'

注意, 单引号或反斜杆(nt: 这里用的是单引号)不能省略, 这可以防止shell对&的解释或替换.

##### ****UDP 数据包****

UDP 数据包的显示格式，可通过rwho这个具体应用所产生的数据包来说明:

actinide.who > broadcast.who: udp 84

其含义为:actinide主机上的端口who向broadcast主机上的端口who发送了一个udp数据包(nt: actinide和broadcast都是指Internet地址).

这个数据包承载的用户数据为84个字节.

一些UDP服务可从数据包的源或目的端口来识别，也可从所显示的更高层协议信息来识别. 比如, Domain Name service requests(DNS 请求,

在RFC-1034/1035中), 和Sun RPC calls to NFS(对NFS服务器所发起的远程调用(nt: 即Sun RPC)，在RFC-1050中有对远程调用的描述).

UDP 名称服务请求

(注意:以下的描述假设你对Domain Service protoco(nt:在RFC-103中有所描述), 否则你会发现以下描述就是天书(nt:希腊文天书,

不必理会, 吓吓你的, 接着看就行))

名称服务请求有如下的格式:

src > dst: id op? flags qtype qclass name (len)

(nt: 从下文来看, 格式应该是src > dst: id op flags qtype qclass? name (len))

比如有一个实际显示为:

h2opolo.1538 > helios.domain: 3+ A? ucbvax.berkeley.edu. (37)

主机h2opolo 向helios 上运行的名称服务器查询ucbvax.berkeley.edu 的地址记录(nt: qtype等于A). 此查询本身的id号为'3'. 符号

'+'意味着递归查询标志被设置(nt: dns服务器可向更高层dns服务器查询本服务器不包含的地址记录). 这个最终通过IP包发送的查询请求

数据长度为37字节, 其中不包括UDP和IP协议的头数据. 因为此查询操作为默认值(nt | rt: normal one的理解), op字段被省略.

如果op字段没被省略, 会被显示在'3' 和'+'之间. 同样, qclass也是默认值, C\_IN, 从而也没被显示, 如果没被忽略, 她会被显示在'A'之后.

异常检查会在方括中显示出附加的域:　如果一个查询同时包含一个回应(nt: 可理解为, 对之前其他一个请求的回应), 并且此回应包含权威或附加记录段,

ancount, nscout, arcount(nt: 具体字段含义需补充) 将被显示为'[na]', '[nn]', '[nau]', 其中n代表合适的计数. 如果包中以下

回应位(比如AA位, RA位, rcode位), 或者字节2或3中任何一个'必须为0'的位被置位(nt: 设置为1), '[b2&3]=x' 将被显示, 其中x表示

头部字节2与字节3进行与操作后的值.

UDP 名称服务应答

对名称服务应答的数据包，tcpdump会有如下的显示格式

src > dst: id op rcode flags a/n/au type class data (len)

比如具体显示如下:

helios.domain > h2opolo.1538: 3 3/3/7 A 128.32.137.3 (273)

helios.domain > h2opolo.1537: 2 NXDomain\* 0/1/0 (97)

第一行表示: helios 对h2opolo 所发送的3号查询请求回应了3条回答记录(nt | rt: answer records), 3条名称服务器记录,

以及7条附加的记录. 第一个回答记录(nt: 3个回答记录中的第一个)类型为A(nt: 表示地址), 其数据为internet地址128.32.137.3.

此回应UDP数据包, 包含273字节的数据(不包含UPD和IP的头部数据). op字段和rcode字段被忽略(nt: op的实际值为Query, rcode, 即

response code的实际值为NoError), 同样被忽略的字段还有class 字段(nt | rt: 其值为C\_IN, 这也是A类型记录默认取值)

第二行表示: helios 对h2opolo 所发送的2号查询请求做了回应. 回应中, rcode编码为NXDomain(nt: 表示不存在的域)), 没有回答记录,

但包含一个名称服务器记录, 不包含权威服务器记录(nt | ck: 从上文来看, 此处的authority records 就是上文中对应的additional

records). '\*'表示权威服务器回答标志被设置(nt: 从而additional records就表示的是authority records).

由于没有回答记录, type, class, data字段都被忽略.

flag字段还有可能出现其他一些字符, 比如'-'(nt: 表示可递归地查询, 即RA 标志没有被设置), '|'(nt: 表示被截断的消息, 即TC 标志

被置位). 如果应答(nt | ct: 可理解为, 包含名称服务应答的UDP数据包, tcpdump知道这类数据包该怎样解析其数据)的'question'段一个条

目(entry)都不包含(nt: 每个条目的含义, 需补充),'[nq]' 会被打印出来.

要注意的是:名称服务器的请求和应答数据量比较大, 而默认的68字节的抓取长度(nt: snaplen, 可理解为tcpdump的一个设置选项)可能不足以抓取

数据包的全部内容. 如果你真的需要仔细查看名称服务器的负载, 可以通过tcpdump 的-s 选项来扩大snaplen值.

##### ****SMB/CIFS 解码****

tcpdump 已可以对SMB/CIFS/NBT相关应用的数据包内容进行解码(nt: 分别为'Server Message Block Common', 'Internet File System'

'在TCP/IP上实现的网络协议NETBIOS的简称'. 这几个服务通常使用UDP的137/138以及TCP的139端口). 原来的对IPX和NetBEUI SMB数据包的

解码能力依然可以被使用(nt: NetBEUI为NETBIOS的增强版本).

tcpdump默认只按照最简约模式对相应数据包进行解码, 如果我们想要详尽的解码信息可以使用其-v 启动选现. 要注意的是, -v 会产生非常详细的信息,

比如对单一的一个SMB数据包, 将产生一屏幕或更多的信息, 所以此选项, 确有需要才使用.

关于SMB数据包格式的信息, 以及每个域的含义可以参看www.cifs.org 或者samba.org 镜像站点的pub/samba/specs/ 目录. linux 上的SMB 补丁

(nt | rt: patch)由 Andrew Tridgell (tridge@samba.org)提供.

NFS 请求和回应

tcpdump对Sun NFS(网络文件系统)请求和回应的UDP数据包有如下格式的打印输出:

src.xid > dst.nfs: len op args

src.nfs > dst.xid: reply stat len op results

以下是一组具体的输出数据

sushi.6709 > wrl.nfs: 112 readlink fh 21,24/10.73165

wrl.nfs > sushi.6709: reply ok 40 readlink "../var"

sushi.201b > wrl.nfs:

144 lookup fh 9,74/4096.6878 "xcolors"

wrl.nfs > sushi.201b:

reply ok 128 lookup fh 9,74/4134.3150

第一行输出表明: 主机sushi向主机wrl发送了一个'交换请求'(nt: transaction), 此请求的id为6709(注意, 主机名字后是交换

请求id号, 而不是源端口号). 此请求数据为112字节, 其中不包括UDP和IP头部的长度. 操作类型为readlink(nt: 即此操作为读符号链接操作),

操作参数为fh 21,24/10.73165(nt: 可按实际运行环境, 解析如下, fd 表示描述的为文件句柄, 21,24 表示此句柄所对应设

备的主/从设备号对, 10表示此句柄所对应的i节点编号(nt:每个文件都会在操作系统中对应一个i节点, 限于unix类系统中),

73165是一个编号(nt: 可理解为标识此请求的一个随机数, 具体含义需补充)).

第二行中, wrl 做了'ok'的回应, 并且在results 字段中返回了sushi想要读的符号连接的真实目录(nt: 即sushi要求读的符号连接其实是一个目录).

第三行表明: sushi 再次请求 wrl 在'fh 9,74/4096.6878'所描述的目录中查找'xcolors'文件. 需要注意的是, 每行所显示的数据含义依赖于其中op字段的

类型(nt: 不同op 所对应args 含义不相同), 其格式遵循NFS 协议, 追求简洁明了.

如果tcpdump 的-v选项(详细打印选项) 被设置, 附加的信息将被显示. 比如:

sushi.1372a > wrl.nfs:

148 read fh 21,11/12.195 8192 bytes @ 24576

wrl.nfs > sushi.1372a:

reply ok 1472 read REG 100664 ids 417/0 sz 29388

(-v 选项一般还会打印出IP头部的TTL, ID， length, 以及fragmentation 域, 但在此例中, 都略过了(nt: 可理解为,简洁起见, 做了删减))

在第一行, sushi 请求wrl 从文件 21,11/12.195(nt: 格式在上面有描述)中, 自偏移24576字节处开始, 读取8192字节数据.

Wrl 回应读取成功; 由于第二行只是回应请求的开头片段, 所以只包含1472字节(其他的数据将在接着的reply片段中到来, 但这些数据包不会再有NFS

头, 甚至UDP头信息也为空(nt: 源和目的应该要有), 这将导致这些片段不能满足过滤条件, 从而没有被打印). -v 选项除了显示文件数据信息, 还会显示

附加显示文件属性信息: file type(文件类型, ''REG'' 表示普通文件), file mode(文件存取模式, 8进制表示的), uid 和gid(nt: 文件属主和

组属主), file size (文件大小).

如果-v 标志被多次重复给出(nt: 如-vv)， tcpdump会显示更加详细的信息.

必须要注意的是, NFS 请求包中数据比较多, 如果tcpdump 的snaplen(nt: 抓取长度) 取太短将不能显示其详细信息. 可使用

'-s 192'来增加snaplen, 这可用以监测NFS应用的网络负载(nt: traffic).

NFS 的回应包并不严格的紧随之前相应的请求包(nt: RPC operation). 从而, tcpdump 会跟踪最近收到的一系列请求包, 再通过其

交换序号(nt: transaction ID)与相应请求包相匹配. 这可能产生一个问题， 如果回应包来得太迟, 超出tcpdump 对相应请求包的跟踪范围,

该回应包将不能被分析.

##### ****AFS 请求和回应****

AFS(nt: Andrew 文件系统, Transarc , 未知, 需补充)请求和回应有如下的答应

src.sport > dst.dport: rx packet-type

src.sport > dst.dport: rx packet-type service call call-name args

src.sport > dst.dport: rx packet-type service reply call-name args

elvis.7001 > pike.afsfs:

rx data fs call rename old fid 536876964/1/1 ".newsrc.new"

new fid 536876964/1/1 ".newsrc"

pike.afsfs > elvis.7001: rx data fs reply rename

在第一行, 主机elvis 向pike 发送了一个RX数据包.

这是一个对于文件服务的请求数据包(nt: RX data packet, 发送数据包 , 可理解为发送包过去, 从而请求对方的服务), 这也是一个RPC

调用的开始(nt: RPC, remote procedure call). 此RPC 请求pike 执行rename(nt: 重命名) 操作, 并指定了相关的参数:

原目录描述符为536876964/1/1, 原文件名为 '.newsrc.new', 新目录描述符为536876964/1/1, 新文件名为 '.newsrc'.

主机pike 对此rename操作的RPC请求作了回应(回应表示rename操作成功, 因为回应的是包含数据内容的包而不是异常包).

一般来说, 所有的'AFS RPC'请求被显示时, 会被冠以一个名字(nt: 即decode, 解码), 这个名字往往就是RPC请求的操作名.

并且, 这些RPC请求的部分参数在显示时, 也会被冠以一个名字(nt | rt: 即decode, 解码, 一般来说也是取名也很直接, 比如,

一个interesting 参数, 显示的时候就会直接是'interesting', 含义拗口, 需再翻).

这种显示格式的设计初衷为'一看就懂', 但对于不熟悉AFS 和 RX 工作原理的人可能不是很

有用(nt: 还是不用管, 书面吓吓你的, 往下看就行).

如果 -v(详细)标志被重复给出(nt: 如-vv), tcpdump 会打印出确认包(nt: 可理解为, 与应答包有区别的包)以及附加头部信息

(nt: 可理解为, 所有包, 而不仅仅是确认包的附加头部信息), 比如, RX call ID(请求包中'请求调用'的ID),

call number('请求调用'的编号), sequence number(nt: 包顺序号),

serial number(nt | rt: 可理解为与包中数据相关的另一个顺信号, 具体含义需补充), 请求包的标识. (nt: 接下来一段为重复描述,

所以略去了), 此外确认包中的MTU协商信息也会被打印出来(nt: 确认包为相对于请求包的确认包, Maximum Transmission Unit, 最大传输单元).

如果 -v 选项被重复了三次(nt: 如-vvv), 那么AFS应用类型数据包的'安全索引'('security index')以及'服务索引'('service id')将会

被打印.

对于表示异常的数据包(nt: abort packet, 可理解为, 此包就是用来通知接受者某种异常已发生), tcpdump 会打印出错误号(error codes).

但对于Ubik beacon packets(nt: Ubik 灯塔指示包, Ubik可理解为特殊的通信协议, beacon packets, 灯塔数据包, 可理解为指明通信中

关键信息的一些数据包), 错误号不会被打印, 因为对于Ubik 协议, 异常数据包不是表示错误, 相反却是表示一种肯定应答(nt: 即, yes vote).

AFS 请求数据量大, 参数也多, 所以要求tcpdump的 snaplen 比较大, 一般可通过启动tcpdump时设置选项'-s 256' 来增大snaplen, 以

监测AFS 应用通信负载.

AFS 回应包并不显示标识RPC 属于何种远程调用. 从而, tcpdump 会跟踪最近一段时间内的请求包, 并通过call number(调用编号), service ID

(服务索引) 来匹配收到的回应包. 如果回应包不是针对最近一段时间内的请求包, tcpdump将无法解析该包.

##### ****KIP AppleTalk协议****

(nt | rt: DDP in UDP可理解为, DDP, The AppleTalk Data Delivery Protocol,

相当于支持KIP AppleTalk协议栈的网络层协议, 而DDP 本身又是通过UDP来传输的,

即在UDP 上实现的用于其他网络的网络层，KIP AppleTalk是苹果公司开发的整套网络协议栈).

AppleTalk DDP 数据包被封装在UDP数据包中, 其解封装(nt: 相当于解码)和相应信息的转储也遵循DDP 包规则.

(nt:encapsulate, 封装, 相当于编码, de-encapsulate, 解封装, 相当于解码, dump, 转储, 通常就是指对其信息进行打印).

/etc/atalk.names 文件中包含了AppleTalk 网络和节点的数字标识到名称的对应关系. 其文件格式通常如下所示:

number name

1.254 ether

16.1 icsd-net

1.254.110 ace

头两行表示有两个AppleTalk 网络. 第三行给出了特定网络上的主机(一个主机会用3个字节来标识,

而一个网络的标识通常只有两个字节, 这也是两者标识的主要区别)(nt: 1.254.110 可理解为ether网络上的ace主机).

标识与其对应的名字之间必须要用空白分开. 除了以上内容, /etc/atalk.names中还包含空行以及注释行(以'#'开始的行).

AppleTalk 完整网络地址将以如下格式显示:

net.host.port

以下为一段具体显示:

144.1.209.2 > icsd-net.112.220

office.2 > icsd-net.112.220

jssmag.149.235 > icsd-net.2

(如果/etc/atalk.names 文件不存在, 或者没有相应AppleTalk 主机/网络的条目, 数据包的网络地址将以数字形式显示).

在第一行中, 网络144.1上的节点209通过2端口,向网络icsd-net上监听在220端口的112节点发送了一个NBP应用数据包

(nt | rt: NBP, name binding protocol, 名称绑定协议, 从数据来看, NBP服务器会在端口2提供此服务.

'DDP port 2' 可理解为'DDP 对应传输层的端口2', DDP本身没有端口的概念, 这点未确定, 需补充).

第二行与第一行类似, 只是源的全部地址可用'office'进行标识.

第三行表示: jssmag网络上的149节点通过235向icsd-net网络上的所有节点的2端口(NBP端口)发送了数据包.(需要注意的是,

在AppleTalk 网络中如果地址中没有节点, 则表示广播地址, 从而节点标识和网络标识最好在/etc/atalk.names有所区别.

nt: 否则一个标识x.port 无法确定x是指一个网络上所有主机的port口还是指定主机x的port口).

tcpdump 可解析NBP (名称绑定协议) and ATP (AppleTalk传输协议)数据包, 对于其他应用层的协议, 只会打印出相应协议名字(

如果此协议没有注册一个通用名字, 只会打印其协议号)以及数据包的大小.

NBP 数据包会按照如下格式显示:

icsd-net.112.220 > jssmag.2: nbp-lkup 190: "=:LaserWriter@\*"

jssmag.209.2 > icsd-net.112.220: nbp-reply 190: "RM1140:LaserWriter@\*" 250

techpit.2 > icsd-net.112.220: nbp-reply 190: "techpit:LaserWriter@\*" 186

第一行表示: 网络icsd-net 中的节点112 通过220端口向网络jssmag 中所有节点的端口2发送了对'LaserWriter'的名称查询请求(nt:

此处名称可理解为一个资源的名称, 比如打印机). 此查询请求的序列号为190.

第二行表示: 网络jssmag 中的节点209 通过2端口向icsd-net.112节点的端口220进行了回应: 我有'LaserWriter'资源, 其资源名称

为'RM1140', 并且在端口250上提供改资源的服务. 此回应的序列号为190, 对应之前查询的序列号.

第三行也是对第一行请求的回应: 节点techpit 通过2端口向icsd-net.112节点的端口220进行了回应:我有'LaserWriter'资源, 其资源名称

为'techpit', 并且在端口186上提供改资源的服务. 此回应的序列号为190, 对应之前查询的序列号.

ATP 数据包的显示格式如下:

jssmag.209.165 > helios.132: atp-req 12266<0-7> 0xae030001

helios.132 > jssmag.209.165: atp-resp 12266:0 (512) 0xae040000

helios.132 > jssmag.209.165: atp-resp 12266:1 (512) 0xae040000

helios.132 > jssmag.209.165: atp-resp 12266:2 (512) 0xae040000

helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000

helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000

helios.132 > jssmag.209.165: atp-resp 12266:6 (512) 0xae040000

helios.132 > jssmag.209.165: atp-resp\*12266:7 (512) 0xae040000

jssmag.209.165 > helios.132: atp-req 12266<3,5> 0xae030001

helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000

helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000

jssmag.209.165 > helios.132: atp-rel 12266<0-7> 0xae030001

jssmag.209.133 > helios.132: atp-req\* 12267<0-7> 0xae030002

第一行表示节点 Jssmag.209 向节点helios 发送了一个会话编号为12266的请求包, 请求helios

回应8个数据包(这8个数据包的顺序号为0-7(nt: 顺序号与会话编号不同, 后者为一次完整传输的编号,

前者为该传输中每个数据包的编号. transaction, 会话, 通常也被叫做传输)). 行尾的16进制数字表示

该请求包中'userdata'域的值(nt: 从下文来看, 这并没有把所有用户数据都打印出来 ).

Helios 回应了8个512字节的数据包. 跟在会话编号(nt: 12266)后的数字表示该数据包在该会话中的顺序号.

括号中的数字表示该数据包中数据的大小, 这不包括atp 的头部. 在顺序号为7数据包(第8行)外带了一个'\*'号,

表示该数据包的EOM 标志被设置了.(nt: EOM, End Of Media, 可理解为, 表示一次会话的数据回应完毕).

接下来的第9行表示, Jssmag.209 又向helios 提出了请求: 顺序号为3以及5的数据包请重新传送. Helios 收到这个

请求后重新发送了这个两个数据包, jssmag.209 再次收到这两个数据包之后, 主动结束(release)了此会话.

在最后一行, jssmag.209 向helios 发送了开始下一次会话的请求包. 请求包中的'\*'表示该包的XO 标志没有被设置.

(nt: XO, exactly once, 可理解为在该会话中, 数据包在接受方只被精确地处理一次, 就算对方重复传送了该数据包,

接收方也只会处理一次, 这需要用到特别设计的数据包接收和处理机制).

##### ****IP 数据包破碎****

(nt: 指把一个IP数据包分成多个IP数据包)

碎片IP数据包(nt: 即一个大的IP数据包破碎后生成的小IP数据包)有如下两种显示格式.

(frag id:size@offset+)

(frag id:size@offset)

(第一种格式表示, 此碎片之后还有后续碎片. 第二种格式表示, 此碎片为最后一个碎片.)

id 表示破碎编号(nt: 从下文来看, 会为每个要破碎的大IP包分配一个破碎编号, 以便区分每个小碎片是否由同一数据包破碎而来).

size 表示此碎片的大小 , 不包含碎片头部数据. offset表示此碎片所含数据在原始整个IP包中的偏移((nt: 从下文来看,

一个IP数据包是作为一个整体被破碎的, 包括头和数据, 而不只是数据被分割).

每个碎片都会使tcpdump产生相应的输出打印. 第一个碎片包含了高层协议的头数据(nt:从下文来看, 被破碎IP数据包中相应tcp头以及

IP头都放在了第一个碎片中 ), 从而tcpdump会针对第一个碎片显示这些信息, 并接着显示此碎片本身的信息. 其后的一些碎片并不包含

高层协议头信息, 从而只会在显示源和目的之后显示碎片本身的信息. 以下有一个例子: 这是一个从arizona.edu 到lbl-rtsg.arpa

途经CSNET网络(nt: CSNET connection 可理解为建立在CSNET 网络上的连接)的ftp应用通信片段:

arizona.ftp-data > rtsg.1170: . 1024:1332(308) ack 1 win 4096 (frag 595a:328@0+)

arizona > rtsg: (frag 595a:204@328)

rtsg.1170 > arizona.ftp-data: . ack 1536 win 2560

有几点值得注意:

第一, 第二行的打印中, 地址后面没有端口号.

这是因为TCP协议信息都放到了第一个碎片中, 当显示第二个碎片时, 我们无法知道此碎片所对应TCP包的顺序号.

第二, 从第一行的信息中, 可以发现arizona需要向rtsg发送308字节的用户数据, 而事实是, 相应IP包经破碎后会总共产生512字节

数据(第一个碎片包含308字节的数据, 第二个碎片包含204个字节的数据, 这超过了308字节). 如果你在查找数据包的顺序号空间中的

一些空洞(nt: hole,空洞, 指数据包之间的顺序号没有上下衔接上), 512这个数据就足够使你迷茫一阵(nt: 其实只要关注308就行,

不必关注破碎后的数据总量).

一个数据包(nt | rt: 指IP数据包)如果带有非IP破碎标志, 则显示时会在最后显示'(DF)'.(nt: 意味着此IP包没有被破碎过).

##### ****时间戳****

tcpdump的所有输出打印行中都会默认包含时间戳信息.

时间戳信息的显示格式如下

hh:mm:ss.frac　(nt: 小时:分钟:秒.(nt: frac未知, 需补充))

此时间戳的精度与内核时间精度一致,　反映的是内核第一次看到对应数据包的时间(nt: saw, 即可对该数据包进行操作).

而数据包从物理线路传递到内核的时间, 以及内核花费在此包上的中断处理时间都没有算进来.

#### 命令使用

tcpdump采用命令行方式，它的命令格式为：

tcpdump [ -AdDeflLnNOpqRStuUvxX ] [ -c count ]

[ -C file\_size ] [ -F file ]

[ -i interface ] [ -m module ] [ -M secret ]

[ -r file ] [ -s snaplen ] [ -T type ] [ -w file ]

[ -W filecount ]

[ -E spi@ipaddr algo:secret,... ]

[ -y datalinktype ] [ -Z user ]

[ expression ]

##### ****tcpdump的简单选项介绍****

-A 以ASCII码方式显示每一个数据包(不会显示数据包中链路层头部信息). 在抓取包含网页数据的数据包时, 可方便查看数据(nt: 即Handy for capturing web pages).

-c count

tcpdump将在接受到count个数据包后退出.

-C file-size (nt: 此选项用于配合-w file 选项使用)

该选项使得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)

-d 以容易阅读的形式,在标准输出上打印出编排过的包匹配码, 随后tcpdump停止.(nt | rt: human readable, 容易阅读的,通常是指以ascii码来打印一些信息. compiled, 编排过的. packet-matching code, 包匹配码,含义未知, 需补充)

-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(会在包匹配码之前有一个附加的'count'前缀).

-D 打印系统中所有tcpdump可以在其上进行抓包的网络接口. 每一个接口会打印出数字编号, 相应的接口名字, 以及可能的一个网络接口描述. 其中网络接口名字和数字编号可以用在tcpdump 的-i flag 选项(nt: 把名字或数字代替flag), 来指定要在其上抓包的网络接口.

此选项在不支持接口列表命令的系统上很有用(nt: 比如, Windows 系统, 或缺乏 ifconfig -a 的UNIX系统); 接口的数字编号在windows 2000 或其后的系统中很有用, 因为这些系统上的接口名字比较复杂, 而不易使用.

如果tcpdump编译时所依赖的libpcap库太老,-D 选项不会被支持, 因为其中缺乏 pcap\_findalldevs()函数.

-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 spi@ipaddr algo:secret,...

可通过spi@ipaddr algo:secret 来解密IPsec ESP包(nt | rt:IPsec Encapsulating Security Payload,IPsec 封装安全负载, IPsec可理解为, 一整套对ip数据包的加密协议, ESP 为整个IP 数据包或其中上层协议部分被加密后的数据,前者的工作模式称为隧道模式; 后者的工作模式称为传输模式 . 工作原理, 另需补充).

需要注意的是, 在终端启动tcpdump 时, 可以为IPv4 ESP packets 设置密钥(secret）.

可用于加密的算法包括des-cbc, 3des-cbc, blowfish-cbc, rc3-cbc, cast128-cbc, 或者没有(none).默认的是des-cbc(nt: des, Data Encryption Standard, 数据加密标准, 加密算法未知, 另需补充).secret 为用于ESP 的密钥, 使用ASCII 字符串方式表达. 如果以 0x 开头, 该密钥将以16进制方式读入.

该选项中ESP 的定义遵循RFC2406, 而不是 RFC1827. 并且, 此选项只是用来调试的, 不推荐以真实密钥(secret)来使用该选项, 因为这样不安全: 在命令行中输入的secret 可以被其他人通过ps 等命令查看到.

除了以上的语法格式(nt: 指spi@ipaddr algo:secret), 还可以在后面添加一个语法输入文件名字供tcpdump 使用(nt：即把spi@ipaddr algo:secret,... 中...换成一个语法文件名). 此文件在接受到第一个ESP　包时会打开此文件, 所以最好此时把赋予tcpdump 的一些特权取消(nt: 可理解为, 这样防范之后, 当该文件为恶意编写时,不至于造成过大损害).

-f 显示外部的IPv4 地址时(nt: foreign IPv4 addresses, 可理解为, 非本机ip地址), 采用数字方式而不是名字.(此选项是用来对付Sun公司的NIS服务器的缺陷(nt: NIS, 网络信息服务, tcpdump 显示外部地址的名字时会用到她提供的名称服务): 此NIS服务器在查询非本地地址名字时,常常会陷入无尽的查询循环).

由于对外部(foreign)IPv4地址的测试需要用到本地网络接口(nt: tcpdump 抓包时用到的接口)及其IPv4 地址和网络掩码. 如果此地址或网络掩码不可用, 或者此接口根本就没有设置相应网络地址和网络掩码(nt: linux 下的 'any' 网络接口就不需要设置地址和掩码, 不过此'any'接口可以收到系统中所有接口的数据包), 该选项不能正常工作.

-F file

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.

-i interface

指定tcpdump 需要监听的接口. 如果没有指定, tcpdump 会从系统接口列表中搜寻编号最小的已配置好的接口(不包括 loopback 接口).一但找到第一个符合条件的接口, 搜寻马上结束.

在采用2.2版本或之后版本内核的Linux 操作系统上, 'any' 这个虚拟网络接口可被用来接收所有网络接口上的数据包(nt: 这会包括目的是该网络接口的, 也包括目的不是该网络接口的). 需要注意的是如果真实网络接口不能工作在'混杂'模式(promiscuous)下,则无法在'any'这个虚拟的网络接口上抓取其数据包.

如果 -D 标志被指定, tcpdump会打印系统中的接口编号，而该编号就可用于此处的interface 参数.

-l 对标准输出进行行缓冲(nt: 使标准输出设备遇到一个换行符就马上把这行的内容打印出来).在需要同时观察抓包打印以及保存抓包记录的时候很有用. 比如, 可通过以下命令组合来达到此目的:

``tcpdump -l | tee dat'' 或者 ``tcpdump -l > dat & tail -f dat''.(nt: 前者使用tee来把tcpdump 的输出同时放到文件dat和标准输出中, 而后者通过重定向操作'>', 把tcpdump的输出放到dat 文件中, 同时通过tail把dat文件中的内容放到标准输出中)

-L 列出指定网络接口所支持的数据链路层的类型后退出.(nt: 指定接口通过-i 来指定)

-m module

通过module 指定的file 装载SMI MIB 模块(nt: SMI，Structure of Management Information, 管理信息结构MIB, Management Information Base, 管理信息库. 可理解为, 这两者用于SNMP(Simple Network Management Protoco)协议数据包的抓取. 具体SNMP 的工作原理未知, 另需补充).

此选项可多次使用, 从而为tcpdump 装载不同的MIB 模块.

-M secret 如果TCP 数据包(TCP segments)有TCP-MD5选项(在RFC 2385有相关描述), 则为其摘要的验证指定一个公共的密钥secret.

-n 不对地址(比如, 主机地址, 端口号)进行数字表示到名字表示的转换.

-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'.

-O 不启用进行包匹配时所用的优化代码. 当怀疑某些bug是由优化代码引起的, 此选项将很有用.

-p 一般情况下, 把网络接口设置为非'混杂'模式. 但必须注意 , 在特殊情况下此网络接口还是会以'混杂'模式来工作； 从而, '-p' 的设与不设, 不能当做以下选现的代名词:'ether host {local-hw-add}' 或 'ether broadcast'(nt: 前者表示只匹配以太网地址为host 的包, 后者表示匹配以太网地址为广播地址的数据包).

-q 快速(也许用'安静'更好?)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.

-R 设定tcpdump 对 ESP/AH 数据包的解析按照 RFC1825而不是RFC1829(nt: AH, 认证头, ESP， 安全负载封装, 这两者会用在IP包的安全传输机制中). 如果此选项被设置, tcpdump 将不会打印出'禁止中继'域(nt: relay prevention field). 另外,由于ESP/AH规范中没有规定ESP/AH数据包必须拥有协议版本号域,所以tcpdump不能从收到的ESP/AH数据包中推导出协议版本号.

-r file

从文件file 中读取包数据. 如果file 字段为 '-' 符号, 则tcpdump 会从标准输入中读取包数据.

-S 打印TCP 数据包的顺序号时, 使用绝对的顺序号, 而不是相对的顺序号.(nt: 相对顺序号可理解为, 相对第一个TCP 包顺序号的差距,比如, 接受方收到第一个数据包的绝对顺序号为232323, 对于后来接收到的第2个,第3个数据包, tcpdump会打印其序列号为1, 2分别表示与第一个数据包的差距为1 和 2. 而如果此时-S 选项被设置, 对于后来接收到的第2个, 第3个数据包会打印出其绝对顺序号:232324, 232325).

-s snaplen

设置tcpdump的数据包抓取长度为snaplen, 如果不设置默认将会是68字节(而支持网络接口分接头(nt: NIT, 上文已有描述,可搜索'网络接口分接头'关键字找到那里)的SunOS系列操作系统中默认的也是最小值是96).68字节对于IP, ICMP(nt: Internet Control Message Protocol,因特网控制报文协议), TCP 以及 UDP 协议的报文已足够, 但对于名称服务(nt: 可理解为dns, nis等服务), NFS服务相关的数据包会产生包截短. 如果产生包截短这种情况, tcpdump的相应打印输出行中会出现''[|proto]''的标志（proto 实际会显示为被截短的数据包的相关协议层次). 需要注意的是, 采用长的抓取长度(nt: snaplen比较大), 会增加包的处理时间, 并且会减少tcpdump 可缓存的数据包的数量， 从而会导致数据包的丢失. 所以, 在能抓取我们想要的包的前提下, 抓取长度越小越好.把snaplen 设置为0 意味着让tcpdump自动选择合适的长度来抓取数据包.

-T type

强制tcpdump按type指定的协议所描述的包结构来分析收到的数据包. 目前已知的type 可取的协议为:

aodv (Ad-hoc On-demand Distance Vector protocol, 按需距离向量路由协议, 在Ad hoc(点对点模式)网络中使用),

cnfp (Cisco NetFlow protocol), rpc(Remote Procedure Call), rtp (Real-Time Applications protocol),

rtcp (Real-Time Applications con-trol protocol), snmp (Simple Network Management Protocol),

tftp (Trivial File Transfer Protocol, 碎文件协议), vat (Visual Audio Tool, 可用于在internet 上进行电

视电话会议的应用层协议), 以及wb (distributed White Board, 可用于网络会议的应用层协议).

-t 在每行输出中不打印时间戳

-tt 不对每行输出的时间进行格式处理(nt: 这种格式一眼可能看不出其含义, 如时间戳打印成1261798315)

-ttt tcpdump 输出时, 每两行打印之间会延迟一个段时间(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

-u 打印出未加密的NFS 句柄(nt: handle可理解为NFS 中使用的文件句柄, 这将包括文件夹和文件夹中的文件)

-U 使得当tcpdump在使用-w 选项时, 其文件写入与包的保存同步.(nt: 即, 当每个数据包被保存时, 它将及时被写入文件中,而不是等文件的输出缓冲已满时才真正写入此文件)

-U 标志在老版本的libcap库(nt: tcpdump 所依赖的报文捕获库)上不起作用, 因为其中缺乏pcap\_cump\_flush()函数.

-v 当分析和打印的时候, 产生详细的输出. 比如, 包的生存时间, 标识, 总长度以及IP包的一些选项. 这也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.

-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(nt: telnet 的SB,SE选项含义未知, 另需补充).

-w 把包数据直接写入文件而不进行分析和打印输出. 这些包数据可在随后通过-r 选项来重新读入并进行分析和打印.

-W filecount

此选项与-C 选项配合使用, 这将限制可打开的文件数目, 并且当文件数据超过这里设置的限制时, 依次循环替代之前的文件, 这相当于一个拥有filecount 个文件的文件缓冲池. 同时, 该选项会使得每个文件名的开头会出现足够多并用来占位的0, 这可以方便这些文件被正确的排序.

-x 当分析和打印时, tcpdump 会打印每个包的头部数据, 同时会以16进制打印出每个包的数据(但不包括连接层的头部).总共打印的数据大小不会超过整个数据包的大小与snaplen 中的最小值. 必须要注意的是, 如果高层协议数据没有snaplen 这么长,并且数据链路层(比如, Ethernet层)有填充数据, 则这些填充数据也会被打印.(nt: so for link layers that pad, 未能衔接理解和翻译, 需补充 )

-xx tcpdump 会打印每个包的头部数据, 同时会以16进制打印出每个包的数据, 其中包括数据链路层的头部.

-X 当分析和打印时, tcpdump 会打印每个包的头部数据, 同时会以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).这对于分析一些新协议的数据包很方便.

-XX 当分析和打印时, tcpdump 会打印每个包的头部数据, 同时会以16进制和ASCII码形式打印出每个包的数据, 其中包括数据链路层的头部.这对于分析一些新协议的数据包很方便.

-y datalinktype

设置tcpdump 只捕获数据链路层协议类型是datalinktype的数据包

-Z user

使tcpdump 放弃自己的超级权限(如果以root用户启动tcpdump, tcpdump将会有超级用户权限), 并把当前tcpdump的用户ID设置为user, 组ID设置为user首要所属组的ID(nt: tcpdump 此处可理解为tcpdump 运行之后对应的进程)

此选项也可在编译的时候被设置为默认打开.(nt: 此时user 的取值未知, 需补充)

##### ****tcpdump条件表达式****

  该表达式用于决定哪些数据包将被打印. 如果不给定条件表达式, 网络上所有被捕获的包都会被打印,否则, 只有满足条件表达式的数据包被打印.(nt: all packets, 可理解为, 所有被指定接口捕获的数据包).

  表达式由一个或多个'表达元'组成(nt: primitive, 表达元, 可理解为组成表达式的基本元素). 一个表达元通常由一个或多个修饰符(qualifiers)后跟一个名字或数字表示的id组成(nt: 即, 'qualifiers id').有三种不同类型的修饰符:type, dir以及 proto.

type 修饰符指定id 所代表的对象类型, id可以是名字也可以是数字. 可选的对象类型有: host, net, port 以及portrange(nt: host 表明id表示主机, net 表明id是网络, port 表明id是端而portrange 表明id 是一个端口范围). 如, 'host foo', 'net 128.3', 'port 20', 'portrange 6000-6008'(nt: 分别表示主机 foo,网络 128.3, 端口 20, 端口范围 6000-6008). 如果不指定type 修饰符, id默认的修饰符为host.

dir 修饰符描述id 所对应的传输方向, 即发往id 还是从id 接收（nt: 而id 到底指什么需要看其前面的type 修饰符）.可取的方向为: src, dst, src 或 dst, src并且dst.(nt:分别表示, id是传输源, id是传输目的, id是传输源或者传输目的, id是传输源并且是传输目的). 例如, 'src foo','dst net 128.3', 'src or dst port ftp-data'.(nt: 分别表示符合条件的数据包中, 源主机是foo, 目的网络是128.3, 源或目的端口为 ftp-data).如果不指定dir修饰符, id 默认的修饰符为src 或 dst.对于链路层的协议,比如SLIP(nt: Serial Line InternetProtocol, 串联线路网际网络协议), 以及linux下指定'any' 设备, 并指定'cooked'(nt | rt: cooked 含义未知, 需补充) 抓取类型, 或其他设备类型,可以用'inbound' 和 'outbount' 修饰符来指定想要的传输方向.

proto 修饰符描述id 所属的协议. 可选的协议有: ether, fddi, tr, wlan, ip, ip6, arp, rarp, decnet, tcp以及 upd.(nt | rt: ether, fddi, tr, 具体含义未知, 需补充. 可理解为物理以太网传输协议, 光纤分布数据网传输协议,以及用于路由跟踪的协议. wlan, 无线局域网协议; ip,ip6 即通常的TCP/IP协议栈中所使用的ipv4以及ipv6网络层协议;arp, rarp 即地址解析协议,反向地址解析协议; decnet, Digital Equipment Corporation开发的, 最早用于PDP-11 机器互联的网络协议; tcp and udp, 即通常TCP/IP协议栈中的两个传输层协议).

例如, `ether src foo', `arp net 128.3', `tcp port 21', `udp portrange 7000-7009'分别表示 '从以太网地址foo 来的数据包','发往或来自128.3网络的arp协议数据包', '发送或接收端口为21的tcp协议数据包', '发送或接收端口范围为7000-7009的udp协议数据包'.

如果不指定proto 修饰符, 则默认为与相应type匹配的修饰符. 例如, 'src foo' 含义是 '(ip or arp or rarp) src foo' (nt: 即, 来自主机foo的ip/arp/rarp协议数据包, 默认type为host),`net bar' 含义是`(ip or arp or rarp) net bar'(nt: 即, 来自或发往bar网络的ip/arp/rarp协议数据包),`port 53' 含义是 `(tcp or udp) port 53'(nt: 即, 发送或接收端口为53的tcp/udp协议数据包).(nt: 由于tcpdump 直接通过数据链路层的 BSD 数据包过滤器或 DLPI(datalink provider interface, 数据链层提供者接口)来直接获得网络数据包, 其可抓取的数据包可涵盖上层的各种协议, 包括arp, rarp, icmp(因特网控制报文协议),ip, ip6, tcp, udp, sctp(流控制传输协议).

对于修饰符后跟id 的格式,可理解为, type id 是对包最基本的过滤条件: 即对包相关的主机, 网络, 端口的限制;dir 表示对包的传送方向的限制; proto表示对包相关的协议限制)

'fddi'(nt: Fiber Distributed Data Interface) 实际上与'ether' 含义一样: tcpdump 会把他们当作一种''指定网络接口上的数据链路层协议''. 如同ehter网(以太网), FDDI 的头部通常也会有源, 目的, 以及包类型, 从而可以像ether网数据包一样对这些域进行过滤. 此外, FDDI 头部还有其他的域, 但不能被放到表达式中用来过滤

同样, 'tr' 和 'wlan' 也和 'ether' 含义一致, 上一段对fddi 的描述同样适用于tr(Token Ring) 和wlan(802.11 wireless LAN)的头部. 对于802.11 协议数据包的头部, 目的域称为DA, 源域称为 SA;而其中的 BSSID, RA, TA 域(nt | rt: 具体含义需补充)不会被检测(nt: 不能被用于包过虑表达式中).

  除以上所描述的表达元('primitive')， 还有其他形式的表达元, 并且与上述表达元格式不同. 比如: gateway, broadcast, less, greater以及算术表达式(nt: 其中每一个都算一种新的表达元). 下面将会对这些表达元进行说明.

  表达元之间还可以通过关键字and, or 以及 not 进行连接, 从而可组成比较复杂的条件表达式. 比如,`host foo and not port ftp and not port ftp-data'(nt: 其过滤条件可理解为, 数据包的主机为foo,并且端口不是ftp(端口21) 和ftp-data(端口20, 常用端口和名字的对应可在linux 系统中的/etc/service 文件中找到)).

  为了表示方便, 同样的修饰符可以被省略, 如'tcp dst port ftp or ftp-data or domain' 与以下的表达式含义相同'tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain'.(nt: 其过滤条件可理解为,包的协议为tcp, 目的端口为ftp 或 ftp-data 或 domain(端口53) ).

  借助括号以及相应操作符,可把表达元组合在一起使用(由于括号是shell的特殊字符, 所以在shell脚本或终端中使用时必须对括号进行转义, 即'(' 与')'需要分别表达成'\(' 与 '\)').

  有效的操作符有:

否定操作 (`!' 或 `not')

与操作(`&&' 或 `and')

或操作(`||' 或 `or')

  否定操作符的优先级别最高. 与操作和或操作优先级别相同, 并且二者的结合顺序是从左到右. 要注意的是, 表达'与操作'时,

  需要显式写出'and'操作符, 而不只是把前后表达元并列放置(nt: 二者中间的'and' 操作符不可省略).

  如果一个标识符前没有关键字, 则表达式的解析过程中最近用过的关键字(往往也是从左往右距离标识符最近的关键字)将被使用.比如,

    not host vs and ace

  是以下表达的精简:

    not host vs and host ace

  而不是not (host vs or ace).(nt: 前两者表示, 所需数据包不是来自或发往host vs, 而是来自或发往ace.而后者表示数据包只要不是来自或发往vs或ac都符合要求)

  整个条件表达式可以被当作一个单独的字符串参数也可以被当作空格分割的多个参数传入tcpdump, 后者更方便些. 通常, 如果表达式中包含元字符(nt: 如正则表达式中的'\*', '.'以及shell中的'('等字符)， 最好还是使用单独字符串的方式传入. 这时,整个表达式需要被单引号括起来. 多参数的传入方式中, 所有参数最终还是被空格串联在一起, 作为一个字符串被解析.

#### 附录:tcpdump的表达元

(nt: True 在以下的描述中含义为: 相应条件表达式中只含有以下所列的一个特定表达元, 此时表达式为真, 即条件得到满足)

dst host host

如果IPv4/v6 数据包的目的域是host, 则与此对应的条件表达式为真.host 可以是一个ip地址, 也可以是一个主机名.

src host host

如果IPv4/v6 数据包的源域是host, 则与此对应的条件表达式为真.

host 可以是一个ip地址, 也可以是一个主机名.

host host

如果IPv4/v6数据包的源或目的地址是 host, 则与此对应的条件表达式为真.以上的几个host 表达式之前可以添加以下关键字:ip, arp, rarp, 以及 ip6.比如:

ip host host

也可以表达为:

ether proto \ip and host host(nt: 这种表达方式在下面有说明, 其中ip之前需要有\来转义,因为ip 对tcpdump 来说已经是一个关键字了.)

如果host 是一个拥有多个IP 的主机, 那么任何一个地址都会用于包的匹配(nt: 即发向host 的数据包的目的地址可以是这几个IP中的任何一个, 从host 接收的数据包的源地址也可以是这几个IP中的任何一个).

ether dst ehost

如果数据包(nt: 指tcpdump 可抓取的数据包, 包括ip 数据包, tcp数据包)的以太网目标地址是ehost,则与此对应的条件表达式为真. Ehost 可以是/etc/ethers 文件中的名字或一个数字地址(nt: 可通过 man ethers 看到对/etc/ethers 文件的描述, 样例中用的是数字地址)

ether src ehost

如果数据包的以太网源地址是ehost, 则与此对应的条件表达式为真.

ether host ehost

如果数据包的以太网源地址或目标地址是ehost, 则与此对应的条件表达式为真.

gateway host

如果数据包的网关地址是host, 则与此对应的条件表达式为真. 需要注意的是, 这里的网关地址是指以太网地址, 而不是IP 地址(nt | rt: I.e., 例如, 可理解为'注意'.the Ethernet source or destination address, 以太网源和目标地址, 可理解为, 指代上句中的'网关地址' ).host 必须是名字而不是数字, 并且必须在机器的'主机名-ip地址'以及'主机名-以太地址'两大映射关系中 有其条目(前一映射关系可通过/etc/hosts文件, DNS 或 NIS得到, 而后一映射关系可通过/etc/ethers 文件得到. nt: /etc/ethers并不一定存在 , 可通过man ethers 看到其数据格式, 如何创建该文件, 未知,需补充).也就是说host 的含义是 ether host ehost 而不是 host host, 并且ehost必须是名字而不是数字.

目前, 该选项在支持IPv6地址格式的配置环境中不起作用(nt: configuration, 配置环境, 可理解为,通信双方的网络配置).

dst net net

如果数据包的目标地址(IPv4或IPv6格式)的网络号字段为 net, 则与此对应的条件表达式为真.

net 可以是从网络数据库文件/etc/networks 中的名字, 也可以是一个数字形式的网络编号.

一个数字IPv4 网络编号将以点分四元组(比如, 192.168.1.0), 或点分三元组(比如, 192.168.1 ), 或点分二元组(比如, 172.16), 或单一单元组(比如, 10)来表达;

对应于这四种情况的网络掩码分别是:四元组:255.255.255.255(这也意味着对net 的匹配如同对主机地址(host)的匹配:地址的四个部分都用到了),三元组:255.255.255.0, 二元组: 255.255.0.0, 一元组:255.0.0.0.

对于IPv6 的地址格式, 网络编号必须全部写出来(8个部分必须全部写出来); 相应网络掩码为:

ff:ff:ff:ff:ff:ff:ff:ff, 所以IPv6 的网络匹配是真正的'host'方式的匹配(nt | rt | rc:地址的8个部分都会用到,是否不属于网络的字节填写0, 需接下来补充), 但同时需要一个网络掩码长度参数来具体指定前面多少字节为网络掩码(nt: 可通过下面的net net/len 来指定)

src net net

如果数据包的源地址(IPv4或IPv6格式)的网络号字段为 net, 则与此对应的条件表达式为真.

net net

如果数据包的源或目的地址(IPv4或IPv6格式)的网络号字段为 net, 则与此对应的条件表达式为真.

net net mask netmask

如果数据包的源或目的地址(IPv4或IPv6格式)的网络掩码与netmask 匹配, 则与此对应的条件表达式为真.此选项之前还可以配合src和dst来匹配源网络地址或目标网络地址(nt: 比如 src net net mask 255.255.255.0).该选项对于ipv6 网络地址无效.

net net/len

如果数据包的源或目的地址(IPv4或IPv6格式)的网络编号字段的比特数与len相同, 则与此对应的条件表达式为真.此选项之前还可以配合src和dst来匹配源网络地址或目标网络地址(nt | rt | tt: src net net/24, 表示需要匹配源地址的网络编号有24位的数据包).

dst port port

如果数据包(包括ip/tcp, ip/udp, ip6/tcp or ip6/udp协议)的目的端口为port, 则与此对应的条件表达式为真.port 可以是一个数字也可以是一个名字(相应名字可以在/etc/services 中找到该名字, 也可以通过man tcp 和man udp来得到相关描述信息 ). 如果使用名字, 则该名字对应的端口号和相应使用的协议都会被检查. 如果只是使用一个数字端口号,则只有相应端口号被检查(比如, dst port 513 将会使tcpdump抓取tcp协议的login 服务和udp协议的who 服务数据包, 而port domain 将会使tcpdump 抓取tcp协议的domain 服务数据包, 以及udp 协议的domain 数据包)(nt | rt: ambiguous name is used 不可理解, 需补充).

src port port

如果数据包的源端口为port, 则与此对应的条件表达式为真.

port port

如果数据包的源或目的端口为port, 则与此对应的条件表达式为真.

dst portrange port1-port2

如果数据包(包括ip/tcp, ip/udp, ip6/tcp or ip6/udp协议)的目的端口属于port1到port2这个端口范围(包括port1, port2), 则与此对应的条件表达式为真. tcpdump 对port1 和port2 解析与对port 的解析一致(nt:在dst port port 选项的描述中有说明).

src portrange port1-port2

如果数据包的源端口属于port1到port2这个端口范围(包括 port1, port2), 则与此对应的条件表达式为真.

portrange port1-port2

如果数据包的源端口或目的端口属于port1到port2这个端口范围(包括 port1, port2), 则与此对应的条件表达式为真.

以上关于port 的选项都可以在其前面添加关键字:tcp 或者udp, 比如:

tcp src port port

这将使tcpdump 只抓取源端口是port 的tcp数据包.

less length

如果数据包的长度比length 小或等于length, 则与此对应的条件表达式为真. 这与'len <= length' 的含义一致.

greater length

如果数据包的长度比length 大或等于length, 则与此对应的条件表达式为真. 这与'len >= length' 的含义一致.

ip proto protocol

如果数据包为ipv4数据包并且其协议类型为protocol, 则与此对应的条件表达式为真.

Protocol 可以是一个数字也可以是名字, 比如:icmp6, igmp, igrp(nt: Interior Gateway Routing Protocol,内部网关路由协议), pim(Protocol Independent Multicast, 独立组播协议, 应用于组播路由器),ah, esp(nt: ah, 认证头, esp 安全负载封装, 这两者会用在IP包的安全传输机制中 ), vrrp(Virtual Router Redundancy Protocol, 虚拟路由器冗余协议), udp, or tcp. 由于tcp , udp 以及icmp是tcpdump 的关键字,所以在这些协议名字之前必须要用\来进行转义(如果在C-shell 中需要用\\来进行转义). 注意此表达元不会把数据包中协议头链中所有协议头内容全部打印出来(nt: 实际上只会打印指定协议的一些头部信息, 比如可以用tcpdump -i eth0 'ip proto \tcp and host 192.168.3.144', 则只打印主机192.168.3.144 发出或接收的数据包中tcp 协议头所包含的信息)

ip6 proto protocol

如果数据包为ipv6数据包并且其协议类型为protocol, 则与此对应的条件表达式为真.

注意此表达元不会把数据包中协议头链中所有协议头内容全部打印出来

ip6 protochain protocol

如果数据包为ipv6数据包并且其协议链中包含类型为protocol协议头, 则与此对应的条件表达式为真. 比如,

ip6 protochain 6

将匹配其协议头链中拥有TCP 协议头的IPv6数据包.此数据包的IPv6头和TCP头之间可能还会包含验证头, 路由头, 或者逐跳寻径选项头.

由此所触发的相应BPF(Berkeley Packets Filter, 可理解为, 在数据链路层提供数据包过滤的一种机制)代码比较繁琐,

并且BPF优化代码也未能照顾到此部分, 从而此选项所触发的包匹配可能会比较慢.

ip protochain protocol

与ip6 protochain protocol 含义相同, 但这用在IPv4数据包.

ether broadcast

如果数据包是以太网广播数据包, 则与此对应的条件表达式为真. ether 关键字是可选的.

ip broadcast

如果数据包是IPv4广播数据包, 则与此对应的条件表达式为真. 这将使tcpdump 检查广播地址是否符合全0和全1的一些约定,并查找网络接口的网络掩码(网络接口为当时在其上抓包的网络接口).

如果抓包所在网络接口的网络掩码不合法, 或者此接口根本就没有设置相应网络地址和网络， 亦或是在linux下的'any'网络接口上抓包(此'any'接口可以收到系统中不止一个接口的数据包(nt: 实际上, 可理解为系统中所有可用的接口)),网络掩码的检查不能正常进行.

ether multicast

如果数据包是一个以太网多点广播数据包(nt: 多点广播, 可理解为把消息同时传递给一组目的地址, 而不是网络中所有地址,后者为可称为广播(broadcast)), 则与此对应的条件表达式为真. 关键字ether 可以省略. 此选项的含义与以下条件表达式含义一致:`ether[0] & 1 != 0'(nt: 可理解为, 以太网数据包中第0个字节的最低位是1, 这意味这是一个多点广播数据包).

ip multicast

如果数据包是ipv4多点广播数据包, 则与此对应的条件表达式为真.

ip6 multicast

如果数据包是ipv6多点广播数据包, 则与此对应的条件表达式为真.

ether proto protocol

如果数据包属于以下以太协议类型, 则与此对应的条件表达式为真.

协议(protocol)字段, 可以是数字或以下所列出了名字: ip, ip6, arp, rarp, atalk(AppleTalk网络协议),

aarp(nt: AppleTalk Address Resolution Protocol, AppleTalk网络的地址解析协议),

decnet(nt: 一个由DEC公司所提供的网络协议栈), sca(nt: 未知, 需补充),

lat(Local Area Transport, 区域传输协议, 由DEC公司开发的以太网主机互联协议),

mopdl, moprc, iso(nt: 未知, 需补充), stp(Spanning tree protocol, 生成树协议, 可用于防止网络中产生链接循环),

ipx（nt: Internetwork Packet Exchange, Novell 网络中使用的网络层协议）, 或者

netbeui(nt: NetBIOS Extended User Interface，可理解为, 网络基本输入输出系统接口扩展).

protocol字段可以是一个数字或以下协议名之一:ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat,

mopdl, moprc, iso, stp, ipx, 或者netbeui.

必须要注意的是标识符也是关键字, 从而必须通过'\'来进行转义.

(SNAP：子网接入协议 （SubNetwork Access Protocol）)

在光纤分布式数据网络接口(其表达元形式可以是'fddi protocol arp'), 令牌环网(其表达元形式可以是'tr protocol arp'),

以及IEEE 802.11 无线局域网(其表达元形式可以是'wlan protocol arp')中, protocol

标识符来自802.2 逻辑链路控制层头,

在FDDI, Token Ring 或 802.1头中会包含此逻辑链路控制层头.

当以这些网络上的相应的协议标识为过滤条件时, tcpdump只是检查LLC头部中以0x000000为组成单元标识符(OUI, 0x000000

标识一个内部以太网)的一段'SNAP格式结构'中的protocol ID 域, 而不会管包中是否有一段OUI为0x000000的'SNAP格式

结构'(nt: SNAP, SubNetwork Access Protocol,子网接入协议 ). 以下例外:

iso tcpdump 会检查LLC头部中的DSAP域(Destination service Access Point, 目标服务接入点)和

SSAP域(源服务接入点).(nt: iso 协议未知, 需补充)

stp 以及 netbeui

tcpdump 将会检查LLC 头部中的目标服务接入点(Destination service Access Point);

atalk

tcpdump 将会检查LLC 头部中以0x080007 为OUI标识的'SNAP格式结构', 并会检查AppleTalk etype域.

(nt: AppleTalk etype 是否位于SNAP格式结构中, 未知, 需补充).

此外, 在以太网中, 对于ether proto protocol 选项, tcpdump 会为 protocol 所指定的协议检查

以太网类型域(the Ethernet type field), 但以下这些协议除外:

iso, stp, and netbeui

tcpdump 将会检查802.3 物理帧以及LLC 头(这两种检查与FDDI, TR, 802.11网络中的相应检查一致);

(nt: 802.3, 理解为IEEE 802.3, 其为一系列IEEE 标准的集合. 此集合定义了有线以太网络中的物理层以及数据

链路层的媒体接入控制子层. stp 在上文已有描述)

atalk

tcpdump 将会检查以太网物理帧中的AppleTalk etype 域 ,　同时也会检查数据包中LLC头部中的'SNAP格式结构'

(这两种检查与FDDI, TR, 802.11网络中的相应检查一致)

aarp tcpdump 将会检查AppleTalk ARP etype 域, 此域或存在于以太网物理帧中, 或存在于LLC(由802.2 所定义)的

'SNAP格式结构'中, 当为后者时, 该'SNAP格式结构'的OUI标识为0x000000;

(nt: 802.2, 可理解为, IEEE802.2, 其中定义了逻辑链路控制层(LLC), 该层对应于OSI 网络模型中数据链路层的上层部分.

LLC 层为使用数据链路层的用户提供了一个统一的接口(通常用户是网络层). LLC层以下是媒体接入控制层(nt: MAC层,

对应于数据链路层的下层部分).该层的实现以及工作方式会根据不同物理传输媒介的不同而有所区别(比如, 以太网, 令牌环网,

光纤分布数据接口(nt: 实际可理解为一种光纤网络), 无线局域网(802.11), 等等.)

ipx tcpdump 将会检查物理以太帧中的IPX etype域, LLC头中的IPX DSAP域，无LLC头并对IPX进行了封装的802.3帧,

以及LLC 头部'SNAP格式结构'中的IPX etype 域(nt | rt: SNAP frame, 可理解为, LLC 头中的'SNAP格式结构'.

该含义属初步理解阶段, 需补充).

decnet src host

如果数据包中DECNET源地址为host, 则与此对应的条件表达式为真.

(nt:decnet, 由Digital Equipment Corporation 开发, 最早用于PDP-11 机器互联的网络协议)

decnet dst host

如果数据包中DECNET目的地址为host, 则与此对应的条件表达式为真.

(nt: decnet 在上文已有说明)

decnet host host

如果数据包中DECNET目的地址或DECNET源地址为host, 则与此对应的条件表达式为真.

(nt: decnet 在上文已有说明)

ifname interface

如果数据包已被标记为从指定的网络接口中接收的, 则与此对应的条件表达式为真.

(此选项只适用于被OpenBSD中pf程序做过标记的包(nt: pf, packet filter, 可理解为OpenBSD中的防火墙程序))

on interface

与 ifname interface 含义一致.

rnr num

如果数据包已被标记为匹配PF的规则, 则与此对应的条件表达式为真.

(此选项只适用于被OpenBSD中pf程序做过标记的包(nt: pf, packet filter, 可理解为OpenBSD中的防火墙程序))

rulenum num

与 rulenum num 含义一致.

reason code

如果数据包已被标记为包含PF的匹配结果代码, 则与此对应的条件表达式为真.有效的结果代码有: match, bad-offset,

fragment, short, normalize, 以及memory.

(此选项只适用于被OpenBSD中pf程序做过标记的包(nt: pf, packet filter, 可理解为OpenBSD中的防火墙程序))

rset name

如果数据包已被标记为匹配指定的规则集, 则与此对应的条件表达式为真.

(此选项只适用于被OpenBSD中pf程序做过标记的包(nt: pf, packet filter, 可理解为OpenBSD中的防火墙程序))

ruleset name

与 rset name 含义一致.

srnr num

如果数据包已被标记为匹配指定的规则集中的特定规则(nt: specified PF rule number, 特定规则编号, 即特定规则),

则与此对应的条件表达式为真.(此选项只适用于被OpenBSD中pf程序做过标记的包(nt: pf, packet filter, 可理解为

OpenBSD中的防火墙程序))

subrulenum num

与 srnr 含义一致.

action act

如果包被记录时PF会执行act指定的动作, 则与此对应的条件表达式为真. 有效的动作有: pass, block.

(此选项只适用于被OpenBSD中pf程序做过标记的包(nt: pf, packet filter, 可理解为OpenBSD中的防火墙程序))

ip, ip6, arp, rarp, atalk, aarp, decnet, iso, stp, ipx, netbeui

与以下表达元含义一致:

ether proto p

p是以上协议中的一个.

lat, moprc, mopdl

与以下表达元含义一致:

ether proto p

p是以上协议中的一个. 必须要注意的是tcpdump目前还不能分析这些协议.

vlan [vlan\_id]

如果数据包为IEEE802.1Q VLAN 数据包, 则与此对应的条件表达式为真.

(nt: IEEE802.1Q VLAN, 即IEEE802.1Q 虚拟网络协议, 此协议用于不同网络的之间的互联).

如果[vlan\_id] 被指定, 则只有数据包含有指定的虚拟网络id(vlan\_id), 则与此对应的条件表达式为真.

要注意的是, 对于VLAN数据包, 在表达式中遇到的第一个vlan关键字会改变表达式中接下来关键字所对应数据包中数据的

开始位置(即解码偏移). 在VLAN网络体系中过滤数据包时, vlan [vlan\_id]表达式可以被多次使用. 关键字vlan每出现一次都会增加

4字节过滤偏移(nt: 过滤偏移, 可理解为上面的解码偏移).

例如:

vlan 100 && vlan 200

表示: 过滤封装在VLAN100中的VLAN200网络上的数据包

再例如:

vlan && vlan 300 && ip

表示: 过滤封装在VLAN300 网络中的IPv4数据包, 而VLAN300网络又被更外层的VLAN封装

mpls [label\_num]

如果数据包为MPLS数据包, 则与此对应的条件表达式为真.

(nt: MPLS, Multi-Protocol Label Switch, 多协议标签交换, 一种在开放的通信网上利用标签引导数据传输的技术).

如果[label\_num] 被指定, 则只有数据包含有指定的标签id(label\_num), 则与此对应的条件表达式为真.

要注意的是, 对于内含MPLS信息的IP数据包(即MPLS数据包), 在表达式中遇到的第一个MPLS关键字会改变表达式中接下来关键字所对应数据包中数据的

开始位置(即解码偏移). 在MPLS网络体系中过滤数据包时, mpls [label\_num]表达式可以被多次使用. 关键字mpls每出现一次都会增加

4字节过滤偏移(nt: 过滤偏移, 可理解为上面的解码偏移).

例如:

mpls 100000 && mpls 1024

表示: 过滤外层标签为100000 而层标签为1024的数据包

再如:

mpls && mpls 1024 && host 192.9.200.1

表示: 过滤发往或来自192.9.200.1的数据包, 该数据包的内层标签为1024, 且拥有一个外层标签.

pppoed

如果数据包为PPP-over-Ethernet的服务器探寻数据包(nt: Discovery packet,

其ethernet type 为0x8863),则与此对应的条件表达式为真.

(nt: PPP-over-Ethernet, 点对点以太网承载协议, 其点对点的连接建立分为Discovery阶段(地址发现) 和

PPPoE 会话建立阶段 , discovery 数据包就是第一阶段发出来的包. ethernet type

是以太帧里的一个字段，用来指明应用于帧数据字段的协议)

pppoes

如果数据包为PPP-over-Ethernet会话数据包(nt: ethernet type 为0x8864, PPP-over-Ethernet在上文已有说明, 可搜索

关键字'PPP-over-Ethernet'找到其描述), 则与此对应的条件表达式为真.

要注意的是, 对于PPP-over-Ethernet会话数据包, 在表达式中遇到的第一个pppoes关键字会改变表达式中接下来关键字所对应数据包中数据的

开始位置(即解码偏移).

例如:

pppoes && ip

表示: 过滤嵌入在PPPoE数据包中的ipv4数据包

tcp, udp, icmp

与以下表达元含义一致:

ip proto p or ip6 proto p

其中p 是以上协议之一(含义分别为: 如果数据包为ipv4或ipv6数据包并且其协议类型为 tcp,udp, 或icmp则与此对

应的条件表达式为真)

iso proto protocol

如果数据包的协议类型为iso-osi协议栈中protocol协议, 则与此对应的条件表达式为真.(nt: [初解]iso-osi 网络模型中每

层的具体协议与tcp/ip相应层采用的协议不同. iso-osi各层中的具体协议另需补充 )

protocol 可以是一个数字编号, 或以下名字中之一:

clnp, esis, or isis.

(nt: clnp, Connectionless Network Protocol, 这是OSI网络模型中网络层协议 , esis, isis 未知, 需补充)

clnp, esis, isis

是以下表达的缩写

iso proto p

其中p 是以上协议之一

l1, l2, iih, lsp, snp, csnp, psnp

为IS-IS PDU 类型 的缩写.

(nt: IS-IS PDU, Intermediate system to intermediate system Protocol Data Unit, 中间系统到

中间系统的协议数据单元. OSI(Open Systems Interconnection)网络由终端系统, 中间系统构成.

终端系统指路由器, 而终端系统指用户设备. 路由器形成的本地组称之为'区域'（Area）和多个区域组成一个'域'（Domain）.

IS-IS 提供域内或区域内的路由. l1, l2, iih, lsp, snp, csnp, psnp 表示PDU的类型, 具体含义另需补充)

vpi n

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 ,

如果数据包为ATM数据包, 并且其虚拟路径标识为n, 则与此对应的条件表达式为真.

(nt: ATM, Asychronous Transfer Mode, 实际上可理解为由ITU-T(国际电信联盟电信标准化部门)提出的一个与

TCP/IP中IP层功能等同的一系列协议, 具体协议层次另需补充)

vci n

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 ,

如果数据包为ATM数据包, 并且其虚拟通道标识为n, 则与此对应的条件表达式为真.

(nt: ATM, 在上文已有描述)

lane

如果数据包为ATM LANE 数据包, 则与此对应的条件表达式为真. 要注意的是, 如果是模拟以太网的LANE数据包或者

LANE逻辑单元控制包, 表达式中第一个lane关键字会改变表达式中随后条件的测试. 如果没有

指定lane关键字, 条件测试将按照数据包中内含LLC(逻辑链路层)的ATM包来进行.

llc

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 ,

如果数据包为ATM数据包,　并且内含LLC则与此对应的条件表达式为真

oamf4s

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是Segment OAM F4 信元(VPI=0 并且 VCI=3), 则与此对应的条件表达式为真.

(nt: OAM, Operation Administration and Maintenance, 操作管理和维护,可理解为:ATM网络中用于网络

管理所产生的ATM信元的分类方式.

ATM网络中传输单位为信元, 要传输的数据终究会被分割成固定长度(53字节)的信元,

(初理解: 一条物理线路可被复用, 形成虚拟路径(virtual path). 而一条虚拟路径再次被复用, 形成虚拟信道(virtual channel)).

通信双方的编址方式为:虚拟路径编号(VPI)/虚拟信道编号(VCI)).

OAM F4 flow 信元又可分为segment 类和end-to-end 类, 其区别未知, 需补充.)

oamf4e

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是 end-to-end OAM F4 信元(VPI=0 并且 VCI=4), 则与此对应的条件表达式为真.

(nt: OAM 与 end-to-end OAM F4 在上文已有描述, 可搜索'oamf4s'来定位)

oamf4

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是 end-to-end 或 segment OAM F4 信元(VPI=0 并且 VCI=3 或者 VCI=4), 则与此对应的条件表达式为真.

(nt: OAM 与 end-to-end OAM F4 在上文已有描述, 可搜索'oamf4s'来定位)

oam

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是 end-to-end 或 segment OAM F4 信元(VPI=0 并且 VCI=3 或者 VCI=4), 则与此对应的条件表达式为真.

(nt: 此选项与oamf4重复, 需确认)

metac

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是来自'元信令线路'(nt: VPI=0 并且 VCI=1, '元信令线路', meta signaling circuit, 具体含义未知, 需补充),

则与此对应的条件表达式为真.

bcc

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是来自'广播信令线路'(nt: VPI=0 并且 VCI=2, '广播信令线路', broadcast signaling circuit, 具体含义未知, 需补充),

则与此对应的条件表达式为真.

sc

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是来自'信令线路'(nt: VPI=0 并且 VCI=5, '信令线路', signaling circuit, 具体含义未知, 需补充),

则与此对应的条件表达式为真.

ilmic

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是来自'ILMI线路'(nt: VPI=0 并且 VCI=16, 'ILMI', Interim Local Management Interface , 可理解为

基于SNMP(简易网络管理协议)的用于网络管理的接口)

则与此对应的条件表达式为真.

connectmsg

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是来自'信令线路'并且是Q.2931协议中规定的以下几种消息: Setup, Calling Proceeding, Connect,

Connect Ack, Release, 或者Release Done. 则与此对应的条件表达式为真.

(nt: Q.2931 为ITU(国际电信联盟)制定的信令协议. 其中规定了在宽带综合业务数字网络的用户接口层建立, 维护, 取消

网络连接的相关步骤.)

metaconnect

如果数据包为ATM数据包, 则与此对应的条件表达式为真. 对于Solaris 操作系统上的SunATM设备 , 如果数据包为ATM数据包

并且是来自'元信令线路'并且是Q.2931协议中规定的以下几种消息: Setup, Calling Proceeding, Connect,

Connect Ack, Release, 或者Release Done. 则与此对应的条件表达式为真.

expr relop expr

如果relop 两侧的操作数(expr)满足relop 指定的关系, 则与此对应的条件表达式为真.

relop 可以是以下关系操作符之一: >, <, <=, =, !=.

expr 是一个算术表达式. 此表达式中可使用整型常量(表示方式与标准C中一致), 二进制操作符(+, -, \*, /, &, |,

<<, >>), 长度操作符, 以及对特定数据包中数据的引用操作符. 要注意的是, 所有的比较操作都默认操作数是无符号的,

例如, 0x80000000 和 0xffffffff 都是大于0的(nt: 对于有符号的比较, 按照补码规则, 0xffffffff

会小于0). 如果要引用数据包中的数据, 可采用以下表达方式:

proto [expr : size]

proto 的取值可以是以下取值之一:ether, fddi, tr, wlan, ppp, slip, link, ip, arp, rarp,

tcp, udp, icmp, ip6 或者 radio. 这指明了该引用操作所对应的协议层.(ether, fddi, wlan,

tr, ppp, slip and link 对应于数据链路层, radio 对应于802.11(wlan,无线局域网)某些数据包中的附带的

"radio"头(nt: 其中描述了波特率, 数据加密等信息)).

要注意的是, tcp, udp 等上层协议目前只能应用于网络层采用为IPv4或IPv6协议的网络(此限制会在tcpdump未来版本中

进行修改). 对于指定协议的所需数据, 其在包数据中的偏移字节由expr 来指定.

以上表达中size 是可选的, 用来指明我们关注那部分数据段的长度(nt:通常这段数据

是数据包的一个域)， 其长度可以是1, 2, 或4个字节. 如果不给定size, 默认是1个字节. 长度操作符的关键字为len,

这代码整个数据包的长度.

例如, 'ether[0] & 1 != 0' 将会使tcpdump 抓取所有多点广播数据包.(nt: ether[0]字节的最低位为1表示

数据包目的地址是多点广播地址). 'ip[0] & 0xf != 5' 对应抓取所有带有选项的

IPv4数据包. 'ip[6:2] & 0x1fff = 0'对应抓取没被破碎的IPv4数据包或者

其片段编号为0的已破碎的IPv4数据包. 这种数据检查方式也适用于tcp和udp数据的引用,

即, tcp[0]对应于TCP 头中第一个字节, 而不是对应任何一个中间的字节.

一些偏移以及域的取值除了可以用数字也可用名字来表达. 以下为可用的一些域(协议头中的域)的名字: icmptype (指ICMP 协议头

中type域), icmpcode (指ICMP 协议头code 域), 以及tcpflags(指TCP协议头的flags 域)

以下为ICMP 协议头中type 域的可用取值:

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.

以下为TCP 协议头中flags 域的可用取值:tcp-fin, tcp-syn, tcp-rst, tcp-push,

tcp-ack, tcp-urg.

### uname

#### SYNOPSIS

uname [option]

#### DESCRIPTION

用来获取电脑和操作系统的相关信息

#### OPTION

Print certain system information. With no OPTION, same as -s.

-a, --all

print all information, in the following order, except omit -p and -i if unknown:

详细输出所有信息，依次为**内核名称**，**主机名**，**内核版本号**，**内核版本**，**硬件名**，**处理器类型**，**硬件平台类型**，**操作系统名称**

-s, --kernel-name

print the kernel name

显示linux内核名称

-n, --nodename

print the network node hostname

显示主机在网络节点上的名称或主机名称

-r, --kernel-release

print the kernel release

显示linux操作系统内核版本号

-v, --kernel-version

print the kernel version

显示显示操作系统是第几个 version 版本

-m, --machine

print the machine hardware name

显示主机的硬件(CPU)名

-p, --processor

print the processor type or "unknown"

显示处理器类型或unknown

-i, --hardware-platform

print the hardware platform or "unknown"

显示硬件平台类型或unknown

-o, --operating-system

print the operating system

显示操作系统名

--help display this help and exit

--version

output version information and exit

#### EXAMPLES

##### 用途

命令

##### 用途

命令

### xargs

大多数 Linux 命令都会产生输出：**文件列表**、**字符串列表**等。但如果要使用其他某个命令并将前一个命令的输出作为参数该怎么办？例如，file 命令显示文件类型（可执行文件、ascii 文本等）；你能处理输出，使其仅显示文件名，目前你希望将这些名称传递给 ls -l 命令以查看时间戳记。xargs 命令就是用来完成此项工作的。他允许你对输出执行其他某些命令。记住下面这个来自于第 1 部分中的语法：

#### file -Lz \* | grep ASCII | cut -d":" -f1 | xargs ls -ltr

让我们来剖析这个命令字符串。第一个，file -Lz \*，用于查找是符号链接或经过压缩的文件。他将输出传递给下一个命令 grep ASCII，该命令在其中搜索 "ASCII" 字符串并产生如下所示的输出： alert\_DBA102.log: ASCII English text

alert\_DBA102.log.Z: ASCII text (compress’d data 16 bits)

dba102\_asmb\_12307.trc.Z: ASCII English text (compress’d data 16 bits)

dba102\_asmb\_20653.trc.Z: ASCII English text (compress’d data 16 bits)

由于我们只对文件名感兴趣，因此我们应用下一个命令 cut -d":" -f1，仅显示第一个字段： alert\_DBA102.log

alert\_DBA102.log.Z

dba102\_asmb\_12307.trc.Z

dba102\_asmb\_20653.trc.Z

目前，我们希望使用 ls -l 命令，将上述列表作为参数进行传递，一次传递一个。xargs 命令允许你这样做。最后一部分，xargs ls -ltr，**用于接收输出并对其执行 ls -ltr 命令**，如下所示：

**ls -ltr alert\_DBA102.log**

**ls -ltr alert\_DBA102.log.Z**

**ls -ltr dba102\_asmb\_12307.trc.Z**

**ls -ltr dba102\_asmb\_20653.trc.Z**

因此，xargs 本身虽然没有多大用处，但在和其他命令相结合时，他的功能非常强大。

下面是另一个示例，我们希望计算这些文件中的行数：

#### $ file \* | grep ASCII | cut -d":" -f1 | xargs wc -l

47853 alert\_DBA102.log

19 dba102\_cjq0\_14493.trc

29053 dba102\_mmnl\_14497.trc

154 dba102\_reco\_14491.trc

43 dba102\_rvwr\_14518.trc

77122 total

（注：上述任务还可用以下命令完成：）

$ wc -l ‘file \* | grep ASCII | cut -d":" -f1 | grep ASCII | cut -d":" -f1‘

该 xargs 版本用于阐释概念。Linux 能用几种方法来完成同一个任务；请使用最适合你的情况的方法。

使用该方法，你能快速重命名目录中的文件。

$ ls | xargs -t -i mv {} {}.bak

-i 选项告诉 xargs 用每项的名称替换 {}。-t 选项指示 xargs 先打印命令，然后再执行。

另一个非常有用的操作是当你使用 vi 打开要编辑的文件时：

$ file \* | grep ASCII | cut -d":" -f1 | xargs vi

该命令使用 vi 逐个打开文件。当你希望搜索多个文件并打开他们进行编辑时，使用该命令非常方便。

他更有几个选项。最有用的可能是 -p 选项，他使操作具有可交互性：

$ file \* | grep ASCII | cut -d":" -f1 | xargs -p vi

vi alert\_DBA102.log dba102\_cjq0\_14493.trc dba102\_mmnl\_14497.trc

dba102\_reco\_14491.trc dba102\_rvwr\_14518.trc ?...

此处的 xarg 需求你在运行每个命令之前进行确认。如果你按下 "y"，则执行命令。当你对文件进行某些可能有破坏且不可恢复的操作（如删除或覆盖）时，你会发现该选项非常有用。

-t 选项使用一个周详模式；他显示要运行的命令，是调试过程中一个非常有帮助的选项。

如果传递给 xargs 的输出为空怎么办？考虑以下命令：

$ file \* | grep SSSSSS | cut -d":" -f1 | xargs -t wc -l

wc -l

0

$

在此处，搜索 "SSSSSS" 后没有匹配的内容；因此 xargs 的输入均为空，如第二行所示（由于我们使用 -t 这个周详选项而产生的结果）。虽然这可能会有所帮助，但在某些情况下，如果没有要处理的内容，你可能希望停止 xargs；如果是这样，能使用 -r 选项： $ file \* | grep SSSSSS | cut -d":" -f1 | xargs -t -r wc -l

$

如果没有要运行的内容，该命令退出。

假设你希望使用 rm 命令（该命令将作为 xargs 命令的参数）删除文件。然而，rm 只能接受有限数量的参数。如果你的参数列表超出该限制怎么办？xargs 的 -n 选项限制单个命令行的参数个数。

下面显示了怎么限制每个命令行仅使用两个参数：即使向 xargs ls -ltr 传递五个文件，但每次向 ls -ltr 仅传递两个文件。

$ file \* | grep ASCII | cut -d":" -f1 | xargs -t -n2 ls -ltr

ls -ltr alert\_DBA102.log dba102\_cjq0\_14493.trc

-rw-r----- 1 oracle dba 738 Aug 10 19:18 dba102\_cjq0\_14493.trc

-rw-r--r-- 1 oracle dba 2410225 Aug 13 05:31 alert\_DBA102.log

ls -ltr dba102\_mmnl\_14497.trc dba102\_reco\_14491.trc

-rw-r----- 1 oracle dba 5386163 Aug 10 17:55 dba102\_mmnl\_14497.trc

-rw-r----- 1 oracle dba 6808 Aug 13 05:21 dba102\_reco\_14491.trc

ls -ltr dba102\_rvwr\_14518.trc

-rw-r----- 1 oracle dba 2087 Aug 10 04:30 dba102\_rvwr\_14518.trc

使用该方法，你能快速重命名目录中的文件。

$ ls | xargs -t -i mv {} {}.bak

-i 选项告诉 xargs 用每项的名称替换 {}。

### zip/unzip

ZIP

zip可能是目前使用得最多的文档压缩格式。它最大的优点就是在不同的操作系统平台，比如Linux， Windows以及Mac OS，上使用。缺点就是支持的压缩率不是很高，而tar.gz和tar.gz2在压缩率方面做得非常好。闲话少说，我们步入正题吧：

我们可以使用下列的**命令压缩一个目录**：

# zip -r archive\_name.zip directory\_to\_compress

下面是如果解压一个zip文档：

# unzip archive\_name.zip

### ln

#### 理解 Linux 的硬链接与软链接

### name

#### SYNOPSIS

tar [option] [file]

#### DESCRIPTION

解压缩文件

GNU `tar' saves many files together into a single tape or disk archive, and can restore individual files from the archive.

#### OPTION

##### –c

--create

create a new archive

建立一个压缩文件的参数指令

注意：不能与x/t共存

##### –x

--extract

extract files from an archive

解开一个压缩文件的参数指令

注意：不能与c/t共存

#### EXAMPLES

##### 用途

命令

##### 用途

命令

## 常用操作

### 文件结束符

Ctrl + D

### 统计代码行数

find ./ "(" -name "\*.cpp" -or -name "\*.h" ")" -print | xargs wc -l

### 修改系统时间

以上两种方法不行的话就采用这样的做法：

1. # cp /etc/localtime /etc/localtime.bak
2. # cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
3. 创建或修改文件 /etc/sysconfig/clock内容：

**ZONE=Asia/Shanghai**

**UTC=false**

**ARC=false**

### 重定向

#### 文件描述符

在linux shell执行命令时，每个进程都和三个打开的文件相联系，并使用文件描述符来引用这些文件。由于文件描述符不容易记忆，shell同时也给出了相应的文件名：

文件 文件描述符

输入文件—标准输入 0（缺省是键盘,为0时是文件或者其他命令的输出）

输出文件—标准输出 1（缺省是屏幕，为1时是文件）

错误输出文件—标准错误 2（缺省是屏幕，为2时是文件） 系统中实际上有12个文件描述符，我们可以任意使用文件描述符3到9.

#### 文件重定向：改变程序运行的输入来源和输出地点

##### 输出重定向

Command > filename 把标准输出重定向到一个新文件中

Command >> filename 把标准输出重定向到一个文件中（追加）

Command > filename 把标准输出重定向到一个文件中

Command > filename 2>&1 把标准输出和错误一起重定向到一个文件中

Command 2> filename 把标准错误重定向到一个文件中

Command 2>> filename 把标准输出重定向到一个文件中（追加）

Command >> filename 2>&1 把标准输出和错误一起重定向到一个文件（追加）

##### 输入重定向

Command < filename > filename2 Command命令以filename文件作为标准输入，以filename2文件作为标准输出

Command < filename Command命令以filename文件作为标准输入

Command << delimiter 从标准输入中读入，知道遇到delimiter分界符

##### 绑定重定向

Command >&m 把标准输出重定向到文件描述符m中

Command < **&-** 关闭标准输入

Command 0>&- 同上

#### shell重定向的一些高级用法

##### 重定向标准错误

例子1：

command 2> /dev/null

如果command执行出错，将错误的信息重定向到空设备

例子2：

command > out.put 2>&1

将command执行的标准输出和标准错误重定向到out.put（也就是说不管command执行正确还是错误，输出都打印到out.put）。

##### exec用法

exec命令可以用来替代当前shell；换句话说，并没有启动子shell，使用这一条命令时任何现有环境变量将会被清除，并重新启动一个shell（重新输入用户名和密码进入）。

exec command

其中，command通常是一个shell脚本。

对文件描述符操作的时候用（也只有再这时候），它不会覆盖你当前的shell

例子1：

#!/bin/bash

#file\_desc

exec 3<&0 0<name.txt

read line1

read line2

exec 0<&3

echo $line1

echo $line2

其中：

首先，exec 3<&0 0<name.txt的意思是把标准输入重定向到文件描述符3（0表示标准输入），然后把文件name.txt内容重定向到文件描述符0，实际上就 是把文件name.txt中的内容重定向到文件描述符3。然后通过exec打开文件描述符3；

然后，通过read命令读取name.txt的第一行内容line1，第二行内容line2，通过Exec 0<&3关闭文件描述符3；

最后，用echo命令输出line1和line2。最好在终端运行一下这个脚本，亲自尝试一下。

例子2：

exec 3<>test.sh;

#打开test.sh可读写操作，与文件描述符3绑定

while read line<&3

do

echo $line;

done

#循环读取文件描述符3（读取的是test.sh内容）

exec 3>&-

exec 3<&-

#关闭文件的，输入，输出绑定

#### 扩展：

在了解重定向之前，我们先来看看linux 的文件描述符。

linux文件描述符：可以理解为linux跟踪打开文件，而分配的一个数字，这个数字有点类似c语言操作文件时候的句柄，通过句柄就可以实现文件的读写操作。 用户可以自定义文件描述符范围是：3-num,这个最大数字，跟用户的：ulimit –n 定义数字有关系，不能超过最大值。

linux启动后，会默认打开3个文件描述符，分别是：标准输入standard input 0,正确输出standard output 1,错误输出：error output 2

以后打开文件后。新增文件绑定描述符 可以依次增加。 一条shell命令执行，都会继承父进程的文件描述符。因此，所有运行的shell命令，都会有默认3个文件描述符。

对于任何一条linux 命令执行，它会是这样一个过程：

image

一个命令执行了：

先有一个输入：输入可以从键盘，也可以从文件得到

命令执行完成：成功了，会把成功结果输出到屏幕：standard output默认是屏幕

命令执行有错误：会把错误也输出到屏幕上面：standard error默认也是指的屏幕

文件输入输出由追踪为一个给定的进程所有打开文件的整数句柄来完成。这些数字值就是文件描述符。最为人们所知的文件米描述符是 stdin, stdout 和 stderr，文件描述符的数字分别是0，1和2。这些数字和各自的设备是保留的。一个命令执行前，先会准备好所有输入输出，默认分别绑定（stdin,stdout,stderr)，如果这个时候出现错误，命令将终止，不会执行。命令解析过程，可以参考：Linux Shell 通配符、元字符、转义符使用实例介绍

这些默认的输出，输入都是linux系统内定的，我们在使用过程中，有时候并不希望执行结果输出到屏幕。我想输出到文件或其它设备。这个时候我们就需要进行输出重定向了。

linux shell下常用输入输出操作符是：

1. 标准输入 (stdin) ：代码为 0 ，使用 < 或 << ； /dev/stdin -> /proc/self/fd/0 0代表：/dev/stdin

2. 标准输出 (stdout)：代码为 1 ，使用 > 或 >> ； /dev/stdout -> /proc/self/fd/1 1代表：/dev/stdout

3. 标准错误输出(stderr)：代码为 2 ，使用 2> 或 2>> ； /dev/stderr -> /proc/self/fd/2 2代表：/dev/stderr

输出重定向：

格式：

command-line1 [1-n] > file或文件操作符或设备

上面命令意思是：将一条命令执行结果（标准输出，或者错误输出，本来都要打印到屏幕上面的） 重定向其它输出设备（文件，打开文件操作符，或打印机等等）1,2分别是标准输出，错误输出。

实例：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

#显示当前目录文件 test.sh test1.sh test1.sh实际不存在

[chengmo@centos5 shell]$ ls test.sh test1.sh

ls: test1.sh: 没有这个文件和目录

test.sh

#正确输出与错误输出都显示在屏幕了，现在需要把正确输出写入suc.txt

# 1>可以省略，不写，默认所至标准输出

[chengmo@centos5 shell]$ ls test.sh test1.sh 1>suc.txt

ls: test1.sh: 没有这个文件和目录

[chengmo@centos5 shell]$ cat suc.txt

test.sh

#把错误输出，不输出到屏幕，输出到err.txt

[chengmo@centos5 shell]$ ls test.sh test1.sh 1>suc.txt 2>err.txt

[chengmo@centos5 shell]$ cat suc.txt err.txt

test.sh

ls: test1.sh: 没有这个文件和目录

#继续追加把输出写入suc.txt err.txt “>>”追加操作符

[chengmo@centos5 shell]$ ls test.sh test1.sh 1>>suc.txt 2>>err.txt

#将错误输出信息关闭掉

[chengmo@centos5 shell]$ ls test.sh test1.sh 2>&-

test.sh

[chengmo@centos5 shell]$ ls test.sh test1.sh 2>/dev/null

test.sh

#&[n] 代表是已经存在的文件描述符，&1 代表输出 &2代表错误输出 &-代表关闭与它绑定的描述符

#/dev/null 这个设备，是linux 中黑洞设备，什么信息只要输出给这个设备，都会给吃掉

#关闭所有输出

[chengmo@centos5 shell]$ ls test.sh test1.sh 1>&- 2>&-

#关闭 1 ，2 文件描述符

[chengmo@centos5 shell]$ ls test.sh test1.sh 2>/dev/null 1>/dev/null

#将1,2 输出转发给/dev/null设备

[chengmo@centos5 shell]$ ls test.sh test1.sh >/dev/null 2>&1

#将错误输出2 绑定给 正确输出 1，然后将 正确输出 发送给 /dev/null设备 这种常用

<p>[chengmo@centos5 shell]$ ls test.sh test1.sh &>/dev/null

#& 代表标准输出 ，错误输出 将所有标准输出与错误输出 输入到/dev/null文件

</p>

注意：

1、shell遇到”>”操作符，会判断右边文件是否存在，如果存在就先删除，并且创建新文件。不存在直接创建。 无论左边命令执行是否成功。右边文件都会变为空。

2、“>>”操作符，判断右边文件，如果不存在，先创建。以添加方式打开文件，会分配一个文件描述符[不特别指定，默认为1,2]然后，与左边的标准输出（1）或错误输出（2） 绑定。

3、当命令：执行完，绑定文件的描述符也自动失效。0,1,2又会空闲。

4、一条命令启动，命令的输入，正确输出，错误输出，默认分别绑定0,1,2文件描述符。

5、一条命令在执行前，先会检查输出是否正确，如果输出设备错误，将不会进行命令执行

输入重定向

格式：

command-line [n] <file或文件描述符&设备

将然有，命令默认从键盘获得的输入，改成从文件，或者其它打开文件以及设备输入。执行这个命令，将标准输入0，与文件或设备绑定。将由它进行输入。

实例：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

[chengmo@centos5 shell]# cat > catfile

testing

cat file test

#这里按下 [ctrl]+d 离开

#从标准输入【键盘】获得数据，然后输出给catfile文件

[chengmo@centos5 shell]$ cat>catfile <test.sh

#cat 从test.sh 获得输入数据，然后输出给文件catfile

[chengmo@centos5 shell]$ cat>catfile <<eof

test a file

test!

eof

#<< 这个连续两个小符号， 他代表的是『结束的输入字符』的意思。这样当空行输入eof字符，输入自动结束，不用ctrl+D

exec绑定重定向

格式：

exec 文件描述符[n] <或> file或文件描述符或设备

在上面讲的输入，输出重定向 将输入，输出绑定文件或设备后。只对当前那条指令是有效的。如果需要在绑定之后，接下来的所有命令都支持的话。就需要用exec命令

实例：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

[chengmo@centos5 shell]$ exec 6>&1

#将标准输出与fd 6绑定

[chengmo@centos5 shell]$ ls /proc/self/fd/

0 1 2 3 6

#出现文件描述符6

[chengmo@centos5 shell]$ exec 1>suc.txt

#将接下来所有命令标准输出，绑定到suc.txt文件（输出到该文件）

[chengmo@centos5 shell]$ ls -al

#执行命令，发现什么都不返回了，因为标准输出已经输出到suc.txt文件了

[chengmo@centos5 shell]$ exec 1>&6

#恢复标准输出

[chengmo@centos5 shell]$ exec 6>&-

#关闭fd 6描述符

[chengmo@centos5 ~]$ ls /proc/self/fd/

0 1 2 3

说明：使用前先将标准输入保存到文件描述符6，这里说明下，文件描述符默认会打开0,1,2 还可以使用自定义描述符 。然后对标准输出绑定到文件，接下来所有输出都会发生到文件。 使用完后，恢复标准的输出，关闭打开文件描述符6。

有趣事情：

可能有朋友会这样用：exec 1>suc.txt ，接下来所有输出都绑定到suc.txt 文件，那么怎么样恢复原来的呢？ 试试你就会发现问题所在……

复杂一点实例

1

2

3

4

5

6

7

8

9

10

11

exec 3<>test.sh;

#打开test.sh可读写操作，与文件描述符3绑定

while read line<&3

do

echo $line;

done

#循环读取文件描述符3（读取的是test.sh内容）

exec 3>&-

exec 3<&-

#关闭文件的，输入，输出绑定

总结下：

学习就要总结，总结才可以提高了。哈哈！

估计还有一些朋友是头晕晕的。怎么linux的重定向这么复杂呢，又是文件打开描述符又是读，还有些，还有默认标准输入输出。

其实，总结一下，重定向应用通常就以下两点：

1、重新设置命令的默认输入，输出，指向到自己文件（文件，文件描述符，设备其实都是文件，因为linux就是基于设备也是文件，描述符也指向是文件，哈哈）

2、扩展自己新的描述符，对文件进行读写操作

### 查看安装的库

1、查看是否安装了gcc

命令 #rpm -qa |grep gcc

#rpm -ql gcc

参数：q-询问

a-查询全部

l-显示列表

### 查看操作系统版本

1. uname –a

Linux localhost.localdomain 3.6.10-4.fc18.i686.PAE #1 SMP Tue Dec 11 18:15:08 UTC 2012 i686 i686 i386 GNU/Linux

1. cat /proc/version

Linux version 3.6.10-4.fc18.i686.PAE (mockbuild@) (gcc version 4.7.2 20121109 (Red Hat 4.7.2-8) (GCC) ) #1 SMP Tue Dec 11 18:15:08 UTC 2012

### 设置网卡IP

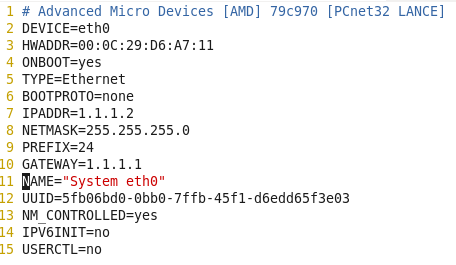
#### 虚拟机网卡配置

1. eth0：网络形式是nat，用于web管理

eth1：网络形式是bridge，用于Styx-fuzzer进行测试

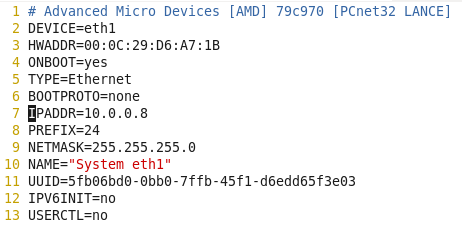
1. vim /etc/sysconfig/network-scripts/ifcfg-eth0

HWADDR需要与ifconfig看到的一致



1. vim /etc/sysconfig/network-scripts/ifcfg-eth1

HWADDR需要与ifconfig看到的一致

****

1. service network restart

#### Styx网卡设置



需求：

* Styx安装在VMware虚拟机里面，两个网卡分别是Eth0和Eth1。其中，Eth0用于访问Styx的WebServer，Eth1用于Styx-fuzzer发送数据包。
* VMware安装在Host上，Host有两个网卡，分别是Connection和Connection2
* 被测设备是Device Under Test。
* 用户在Styx Browser Host上用浏览器访问Styx的WebServer，并用Styx-fuzzer发送数据包测试被测设备。

配置：

* 将VMware的两个VMnet分别与两个网卡进行映射
  + 点击VMware Workstation的菜单Edit🡪Virtual Network Editor...

将VMnet0与Connection2网卡进行映射

### 设置环境变量

#### 直接用export命令

[见export命令示例](#_修改环境变量)

#### 修改profile文件

注意：需要重新注销系统才能生效

* #vi /etc/profile

在里面加入:

export PATH=$PATH:your path

* # reboot
* 测试

# echo $PATH

##### 两个profile文件

* /etc/profile

对所有用户有效

* ~/.bash\_profile

只对当前用户有效（~指的是当前用户主目录）

#### 修改.bashrc文件

注意：需要重新注销系统才能生效

* # vi /root/.bashrc

在里面加入：

export PATH="$PATH:your path "

* # reboot
* 测试

# echo $PATH

### 查找含有关键字的文件

* **一个文件可以显示多次：**

find path/ -name “\*.cpp” –type f –exec grep “” –nH {} \;

* **一个文件只显示一次：**

find path/ -name “\*.cpp” –type f –exec grep “” –nHl {} \;

### 解压缩

zip文件查看zip/unzip命令

其他文件查看tar命令

### tcpdump抓包

使用格式：

tcpdump -U -w file1.pcap -i eth0 host 10.0.0.91 2>/dev/null &

其中，

* -U：与-w配合使用，确保文件写入与包的保存同步，而不是等文件输出缓冲区满了才写入文件（OpenBSD上没有这个功能）。
* -w：写入的文件名
* -i：监听的网卡名
* host 10.0.0.91：在这里代表了过滤规则
* 2>/dev/null：在这里表示将输出到stderr信息过滤掉
* &：表示后台启动这个程序

实时显示捕获到的数据：

使用格式：tcpdump

### Fedora18分辨率

修改grub.cfg。

#sudo gedit /boot/grub2/grub.cfg

当然是修改显示器分辨率那项的

set gfxmode=auto(默认是这样的)，改成

set gfxmode=1440x900【乘号可以写成x】（根据自己显示器设置）就可以了。

### 字符串替换

**sed -i "s/zhangsan/lisi/g" `grep zhangsan -rl /modules`**

1. grep的结果是sed的输入
2. 解释一下：

-i 表示inplace edit，就地修改文件

-r 表示搜索子目录

-l 表示输出匹配的文件名

这个命令组合很强大，要注意备份文件。

（1）sed 'y/1234567890/ABCDEFGHIJ/' test\_sed

sed 'y/1234567890/ABCDEFGHIJ/' filename

ABCDEFGHIJ

BCDEFGHIJA

CDEFGHIJAB

DEFGHIJABC

注意变换关系是按两个list的位置对应变换

其中：test\_sed的内容是：

1234567890

2345678901

3456789012

4567890123

(2)替换每行所有匹配

sed 's/01/Ab/g' test\_sed

1234567890

23456789Ab

3456789Ab2

456789Ab23

注意：第一行的0，1没有分别替换为A,b

## 常用工具

### python3.3.2的安装

见《赵凯Python学习笔记》

### perl模块安装

1. **在cpan网上下载**

ExtUtils-MakeMaker-6.84.tar.gz

ExtUtils-CBuilder-0.280212.tar.gz

1. **安装**

先安装ExtUtils-MakeMaker，因为CBuilder依赖之。

* 1. tar -xzvf ExtUtils-MakeMaker-6.84.tar.gz
  2. cd ExtUtils-MakeMaker-6.84
  3. perl Makefile.Pl
  4. make
  5. make test
  6. make install
  7. 安装CBuilder同上

### net-snmp

#### 安装

* 解压
* 查看READEME.snmpv3
  + 查看snmplib/openssl/READE

编译时使用—with-openssl=internal

* 查看READEME
  + 查看INSTALL

具体步骤（参考网络）：

1. #rpm qa | grep ssl //确认是否安装ssl
2. #tar -zxvf net-snmp-5.4.1.2.tar.gz //解压
3. #cd net-snmp-5.4.1.2 //进入源文件目录
4. #./configure //配置 在过程中需要选择
   1. default version of-snmp-version:3
   2. Systemcontact information（配置该设备的联系人）: zhaokai
   3. System location (该设备的位置):china
   4. Location to write logfile (日志文件位置): 默认直接回车就行
   5. Location to Write persistent(数据存储目录):默认直接回车就行

配置完之后显示如下：

SNMP Versions Supported: 1 2c 3

Building for: linux

Net-SNMP Version: 5.7.1

Network transport support: Callback Unix Alias TCP UDP IPv4Base SocketBase TCPBase UDPIPv4Base UDPBase

SNMPv3 Security Modules: usm

Agent MIB code: default\_modules => snmpv3mibs mibII ucd\_snmp notification notification-log-mib target agent\_mibs agentx disman/event disman/schedule utilities host

MYSQL Trap Logging: unavailable

Embedded Perl support: disabled

SNMP Perl modules: building -- not embeddable

SNMP Python modules: disabled

Crypto support from: crypto

Authentication support: MD5 SHA1

Encryption support: DES AES

Local DNSSEC validation: disabled

1. #make
   1. make出现错误

Can't locate ExtUtils/MakeMaker.pm in @INC (@INC contains: /usr/local/lib/perl5 /usr/local/share/perl5 /usr/lib/perl5/vendor\_perl /usr/share/perl5/vendor\_perl /usr/lib/perl5 /usr/share/perl5 .) at Makefile.PL line 1.

BEGIN failed--compilation aborted at Makefile.PL line 1.

**问题原因：**

perl缺少模块

**解决方案：**

安装perl模块，见perl模块安装

1. #make install
   1. 最后报错退出，没有找到per
2. #安装失败