The truth about processes

A process has multiple userIDs and groupIDs:

- real user ID
  - User who started the process; its "owner"
  - C system call: `getuid()` to obtain this
- real group ID
  - Current group of user who started the process
  - C system call: `getgid()` to obtain this
- effective user ID
  - User ID to use for file permissions
  - *Usually* the same as the real user ID
  - C system call: `geteuid()` to obtain this
- effective group ID
  - Group ID to use for file permissions
  - *Usually* the same as the group user ID
  - C system call: `getegid()` to obtain this

Fun setuid example

```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout
```

Fun setuid example

```
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ cd /tmp
prompt$ cat file.txt
cat: file.txt: Permission denied
prompt$ ./bobcat file.txt
This is an unreadable file
prompt$
```

`authselect` : configure authentication sources
`ldd` : print shared library dependencies
`nisdomainname` : show the NIS domain name
`rpcinfo` : show RPC programs (`-p` for port numbers)
`ypcat` : show an NIS database
`ypmatch` : show a matching key in an NIS database
`yppasswd` : change passwords on an NIS client

Single user mode

Must do all of the following to prevent crackers from booting in single user mode:

1. Use a GrUB password
   - Prevents editing of boot entries
2. Disable "boot from CD drive" in the BIOS
   - Prevents booting a live CD
3. Use a BIOS password
4. Lock machine(s) shut
   - Prevents crackers from resetting BIOS
   - Prevents crackers from changing drives

Intrusion detection

Suppose you *suspect* that someone has compromised your ma (obtained `root` access). How do you check this?

- Look for new accounts in `/etc/passwd`
  - Maybe `cat /etc/passwd` and inspect
  - Or maybe `grep -v nologin /etc/passwd`
- Look for new files
  - Maybe `ls /home`
- Look for unknown running processes
  - Maybe `ps aux | grep -v root`
- Look in system logs (e.g., `/var/log/secure`)

But...

1. How do you know where to look?
2. Why should system logs still be intact?
3. Why should `cat`, `grep`, `ls`, `ps` still work?

Quick example with `firewall-cmd`

```
prompt$ firewall-cmd --get-active-zones
home
  interfaces: eth1
external
  interfaces: eth0
prompt$ firewall-cmd --info-zone=external --permanent
```


Quick example with `firewall-cmd`

```
prompt$ firewall-cmd --info-zone=external --permanent
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

nmap utility

- Port scanner — shows what ports are open on a host
- Useful tool when securing a machine
- Also useful for crackers to see potential entry points
  - So some systems do not take kindly to being scanned

```
prompt$ nmap localhost

Starting Nmap 5.50 ( http://nmap.org ) at 2012-12-05 10:06 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000015s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 997 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind

Nmap done: 1 IP address (1 host up) scanned in 0.15 seconds
prompt$
```

"Backward" quotes

- Are completely different from single and double quotes
- Use the accent character (usually, on your tilde key)
- Usage:

```
`command`
```

- Replaces the string with the output of the command
  - Whatever would have been written to standard output

```
prompt$ today=`date | head -c 10`
prompt$ echo $today
Fri Sep 23
prompt$ echo "The current time is `date | tail -c +11 | head -c 9`"
The current time is  12:03:27
prompt$
```

export example

```
prompt$ FOO="fu"
prompt$ export BAR="bar"
prompt$ echo "$FOO actions mangle systems $BAR"
fu actions mangle systems bar
prompt$ bash
prompt$ echo "$FOO actions mangle systems $BAR"
 actions mangle systems bar
prompt$ BAR="bartastic"
prompt$ exit
prompt$ echo "This example is $FOO$BAR."
This example is fubar.
prompt$ export
declare -x BAR="bar"
prompt$
```

Useful `ls` switches

`-a` : show all files
- Filenames starting with `.` are normally hidden

`-l` : long listing
- Default is "short listing": ordered names in columns

`-F` : extra character displayed for the file type
- `*` : executables
- `/` : subdirectory
- `@` : symlink (will discuss later)
- `=` : socket (discussed in ComS 352)

`--color` : Like "-F" but uses color (Linux only)

`-G` : Like "-F" but uses color (Mac OS only)

System manual

`man` : for "manual"
- Gives the manual pages about a command, utility, configuration file, system call, etc.
- Use: give the commands you want to know about, as arguments.
- To learn about the C function, `fopen`:

```
prompt$ man fopen
```

- To learn about `man` itself:

```
prompt$ man man
```

`info` : like `man`, but fancier
- Opens manual page in a viewer
- Has hyperlinks

Shell script: code2

```
#!/bin/bash
strange()
{
  echo "Strange function"
  return 42
}

ret=$(strange)
echo "Return code: $?"
echo "Got: $ret"
exit 3
```

```
Got: Strange function
prompt$ echo $?
3
prompt$
```

How can we match *all* text of the form:

1. Single–character variable name
2. Space
3. Arithmetic operator
4. Space
5. Single–character variable name

Things like: $x + y$, $I - J$, ...
Use regex: "`[a-zA-Z] [-+*/%] [a-zA-Z]`"

`-eq` : Check numerical equality (replaces =)
- More precisely: "`test str1 -eq str2`" exists successfully if and only if strings `str1` and `str2` are equal when interpreted as numbers

`-ne` : Check numerical inequality (replaces !=)

`-lt` : Check numerical less than (replaces <)

`-le` : Check numerical less or equal (replaces <=)

`-gt` : Check numerical greater than (replaces >)

`-ge` : Check numerical greater or equal (replaces >=)

- To match lines that start with "a":
  `grep '^a'`
- To match lines that end with "a":
  `grep 'a$'`
- To match the line "a":
  `grep '^a$'`
- To match a line containing "^":
  `grep '\^'`
- To match a line containing "$":
  `grep '\$'`

`curl` : copy a URL
`ping` : send IP packets to a machine
`traceroute` : determine a route to a machine

Use "–" inside [] to specify a character range

- E.g., use regex "`[01]`" to match a binary digit
- E.g., use regex "`[0-7]`" to match an octal digit
- E.g., use regex "`[0-9]`" to match a decimal digit
- E.g., use regex "`[0-9a-f]`" to match a hexadecimal digit

Use "–" first or last inside [] to specify "–"

- E.g., to match arithmetic operators, use "`[-+*/]`"

- Does string '7' match regex '`[0-9]*[02468]`'?
  - '7' matches '`[0-9]*`' but '' does not match '`[02468]`'
  - '' matches '`[0-9]*`' but '7' does not match '`[02468]`'

  No.
- Does string '8' match regex '`[0-9]*[02468]`'?
  - '8' matches '`[0-9]*`' but '' does not match '`[02468]`'
  - '' matches '`[0-9]*`' and '8' matches '`[02468]`'

  Yes.
- Does string '84' match regex '`[0-9]*[02468]`'?
  - '8' matches '`[0-9]*`' and '4' matches '`[02468]`'
  - '' matches '`[0-9]*`' and '8' matches '`[02468]`'

    but then we have an extra '4' that matches nothing

  Yes.

Remember: regular expressions match as much as they can

Regular expressions may be grouped using ()

- The "previous thing" for a repeat operator may be a regex
- We can repeat entire subexpressions
- We can make entire subexpressions optional

Simple example:

`ab+` means: 'a' followed by one or more 'b's
- Strings 'ab', 'abb', 'abbb', ..., match this

`(ab)+` means: a sequence of one or more 'ab's
- Strings 'ab', 'abab', 'ababab', ..., match this

`(ab+)+` means:
- One or more strings matching `ab+`, concatenated
- Start with 'a', end with 'b', never 2 'a's together

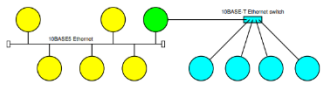
AWK assignment operators

Includes the usual C / Java operators

`=` Assignment
`+=` Add value to a variable
`-=` Subtract value from a variable
`*=` Multiply variable by a value
`/=` Divide variable by a value
`%=` "Mod" variable by a value
`^=` "Exponentiate" variable by a value
`**=` "Exponentiate" variable by a value
- Not POSIX compliant

`++` Increment operator (pre or post)
`--` Decrement operator (pre or post)

Network with 2 subnetworks



The green machine:
- Belongs to both LANs
- Has two network connections
  - Each will have its own name
- Is a known as a gateway
  - Because it is an entrance to another network

Example: show files owned by bob

With a byte total at the end

We will use

```
ls -l | ./bobtotal.awk
```

Need to write the AWK program `bobtotal.awk`

bobtotal.awk

```
#!/usr/bin/awk -f
BEGIN  { total = 0 }
$3 == "bob"  { print; total += $5 }
END    { print "Total is " total " bytes." }
```

Consider the following program:

2.sed

```
#!/usr/bin/sed -f
s/ first/second/
/second/s/These/Those/
```

What does each statement do?

1. On every line, replace " first" with "second"
2. On lines containing "second", replace "These" with "Those"

Let's run this and see what happens...

Example for `nmcli`

- Network device currently using DHCP
- Needs to be set statically
  - IP address and subnet is 10.3.3.3/24
  - Gateway address is 10.3.3.1

```
$ nmcli
enp0s3:  connecting (getting IP configuration) to enp0s3
         "Intel 82540EM"
         ethernet (e1000), 08:00:27:AE:E4:B3, hw, mtu 15

lo:  unmanaged
         "lo"
         loopback (unknown), 00:00:00:00:00:00, sw, mtu 65536
$ nmcli dev
DEVICE  TYPE      STATE         CONNECTION
enp0s3  ethernet  disconnected  --
lo      loopback  unmanaged     --
$ nmcli connecti
```

16–bit subnet prefix example

Subnet mask:	255.255.0.0	= 11111111.11111111.00000000.00000000
Network:	129.186.0.0	= 10000001.10111010.00000000.00000000
First usable:	129.186.0.1	= 10000001.10111010.00000000.00000001
Last usable:	129.186.255.254	= 10000001.10111010.11111111.11111110
Broadcast:	129.186.255.255	= 10000001.10111010.11111111.11111111

24–bit subnet prefix example

Subnet mask:	255.255.255.0	= 11111111.11111111.11111111.00000000
Network:	192.168.2.0	= 11000000.10101000.00000010.00000000
First usable:	192.168.2.1	= 11000000.10101000.00000010.00000001
Last usable:	192.168.2.254	= 11000000.10101000.00000010.11111110
Broadcast:	192.168.2.255	= 11000000.10101000.00000010.11111111

28–bit subnet prefix example

Subnet mask:	255.255.255.240	= 11111111.11111111.11111111.11110000
Network:	172.1.1.176	= 10101100.00000001.00000001.10110000
First usable:	172.1.1.177	= 10101100.00000001.00000001.10110001
Last usable:	172.1.1.190	= 10101100.00000001.00000001.10111110
Broadcast:	172.1.1.191	= 10101100.00000001.00000001.10111111

Generating an ssh key

1. Run `ssh-keygen` to generate a key
   - Can associate a passphrase with the key
   - Key will have two parts
     - (i) A private part (keep this secret)
     - (ii) A public part
   - How it works is quite technical, but:
     - The public key is used to encrypt messages
     - Only the private key can decrypt the message
     - You cannot determine the private key from its public one
2. Add the public key to the appropriate file on the remote host
   - Usually `~/.ssh/authorized_keys`
3. You should be able to `ssh` into the remote host using the key
   - You will be prompted for the key's passphrase
   - You will not be prompted for the password on the remote host

`firewall-cmd` example

```
prompt$ firewall-cmd --add-service=ssh
success
prompt$ firewall-cmd --list-services
http ssh
prompt$ firewall-cmd --remove-service=http
success
prompt$ firewall-cmd --list-services
ssh
prompt$ firewall-cmd --list-services --permanent
mdns dhcpv6-client http
prompt$ firewall-cmd --add-service=ssh --permanent
success
prompt$
```