

Course: CIS 68C1-01

Homework: #3

Assigned: Monday, October 9<sup>th</sup>

Due: Monday, October 15<sup>th</sup>, 6pm

---

1. How in Linux can you force users to change their login passwords every 30 days?

The fifth field in the `/etc/shadow` file is the maximum number of days a user may retain a password before the system will require the user to change it. Setting this field to 30 would force the user to reset the password after 30 days since the last change. See page 656 about issues related forcing users to change passwords.

2. What is wrong with the login name "sam spade"?

Login names may *not* contain spaces, and should be 8 characters or less. The login **sam spade** violates both of these rules. The login **samspade** would be acceptable.

3. Why is it not good practice to assign the same UID to more than one user account?

Assigning the same UID to more than one account removes traceability and security. Files have an owner, which is the UID of the user who created the file. It would be impossible to tell if User A or User B created the file if both users had the same UID. Also, it would be impossible to prevent User A from accessing User B's files.

4. What is the purpose of the `/etc/shadow` file?

The `/etc/shadow` file is used primarily to store passwords for user accounts. This was an added security measure when it was found that by having read access to the `/etc/passwd` file allowed password crackers to crack passwords by comparing the encrypted passwords in `/etc/passwd` with those generated by cracking programs. By removing the passwords from the `/etc/passwd` file, and placing them in a non-readable file such as `/etc/shadow`, this removed the ability for crackers to compare passwords. The only way to crack passwords now is to actually try to login as the user. The login process has been artificially and severely slowed down using delays to prevent cracking passwords easily. In 1998, it was found that a 56-bit *key* could be cracked in a couple of days, and with a super computer, in a matter of hours. And older UNIX password systems do not use this level of security. Linux and other UNIXes now provide support for 56-bit DES encrypted passwords.

The `/etc/shadow` file has additional features such as password aging, and account expiration.

5. What should the file permissions be for the `/etc/shadow` file? For the `/etc/passwd` file?

The permissions on the `/etc/shadow` file should be 600, and on `/etc/passwd` 644. These files should never be writable by group or others, and the shadow file should only be readable by its owner; both files should always be owned by root, with group root.

6. What is wrong with the `/etc/passwd` line below?

```
joeuser:x:987:100:1234:Joe User:/bin/sh:/home/joeuser
```

This `/etc/passwd` entry has two problems. First, it contains 8 fields instead of 7 (colons separate each field). Second, the home directory and shell fields are backwards; the shell field is field #7.

7. Can the program `/bin/date` be used as a login shell? If so, what would happen?

You can use programs such as **date** as a login shell. This would provide an *account* that would simply run the `date` program upon logging into that account, and then it would exit, returning the user back to the `getty` login prompt. It used to be common practice to create several pseudo-accounts such as this, but is not discouraged for security reasons. One such account was the *who* account, that would run the **who** program to tell you who was on a system. This would allow a user to type **who** at the login prompt to see who was on the system. This is not a very good idea in today's inter-connected computing world.

8. Can a user belong to more than one group in the `/etc/group` file?

Yes. On most systems, a user can be a member of up to 16 different groups. The purpose of the `/etc/group` is to support this.

9. Write a line from `/etc/group` that defines a new group named **project**, with GID **500**, and has members **joe**, **sally**, **federico**, and **jane**. Be sure that your syntax is correct.

```
project:x:500:joe,sally,federico,jane
```

10. What should the permissions, UID and GID be for a user's home directory?

Administrators should usually set the permissions of a user's home directory to **700**, which allows the **user** to read, write, and examine the directory, but prevents **group** and **others** from having access. If desired, users can later relax this level of security on their own. A user ID of the user's home directory should *always* be the user's UID. Likewise, the group ID of the directory should be the GID of the user, as set in the `/etc/passwd` file.

11. What startup files might you create for a user whose default shell is `/bin/csh`? What about for a default shell of `/bin/sh`?

The **csh** shell uses **.cshrc** and **.login**, while the **sh** shell uses **.profile**. Administrators should create default template files that are copied to a new user's account when the account is created. Copy the appropriate template files depending on the login shell given to the user, or copy all of them if users can change their login shell.

Additionally, there are global startup files in `/etc` that are **sourced** when a shell is started or upon login. These too should be customized for each site.

12. You've just created a new account for **benstein**, and want to change ownership of **benstein**'s startup files so that **benstein** owns them. What is wrong with the following command? Give a better command.

```
chown -R benstein /home/benstein/.*
```

This is a very common mistake. The `.*` wildcard will match not only the dotfiles, but also the parent directory `..` which in this case is `/home`. Thus, this recursive **chown** will change the owner of *everything* under `/home` and downwards to be **benstein**. This is probably not the desired result. Instead, use `.*?*` or specifically name the files and directories in `/home/benstein`.

13. Name two ways to *disable* a user's account.

To disable an account, place a `*` in the password field of `/etc/passwd` or `/etc/shadow`. A better solution is to change the user's login shell in `/etc/passwd` to be a program that only prints out a message such as *Account Disabled - Contact the system administrator at XXX-XXXX for more information* and then exits. This allows the user to learn why the account is disabled and who to contact if necessary to resolve the problem. Removing or disabling accounts without advanced notice or contact information is considered bad form.

14. What is the most common way that the INT signal is generated?

The most common way is with the `^C` key. The `^C` key is usually *bound* to the interrupt key of the controlling terminal. When pressed, the controlling terminal driver will send the INT signal to the process that currently is attached to the controlling terminal (the currently running or foreground process). The controlling terminal key bindings can be changed with the **stty** command.

15. What happens if you send a TERM signal to a process that is in the Stopped state? Does it terminate? Assume the default action for this signal. Explain your answer thoroughly.

Sending a TERM signal to an already stopped process causes the process to receive the TERM signal only after it has been continued with a CONT signal. The program then has the opportunity to handle the signal and do any necessary cleanup. The shells have a built-in **kill** command which will automatically send a CONT signal before sending the TERM signal to any stopped process. This allows the process to resume and then receive the TERM signal. In most cases, it appears to the user that the process terminates immediately. There seems to be a bug in the **ksh** on Solaris. The process is continued, but the TERM signal is not delivered. Using the **jobs** command shows the process as **running**. However, foregrounding the job causes the session to hang.

16. What happens if you send a KILL signal to a process that is in the Stopped state? Does it terminate? Explain your answer thoroughly.

The kernel does not deliver KILL signals to a process - it terminates the process immediately, regardless of whether it is running or stopped.

17. When a process exits, what mechanism is used so the parent receives the child's exit status?

When process exits, it returns its exit status to its parent using the `exit()` system call. The parent does a `wait()` system call to receive its child's status.

18. What does the command below do?

```
kill -9 15 9 3298
```

This command sends signal number **9** to the three processes with PIDs 15, 9, and 3298. This signal is unblock-able; the kernel will kill these processes immediately.

19. How do you eliminate a Zombie process?

You cannot eliminate a zombie process. A parent should do a `wait()` to receive the child's status, which causes the kernel to remove the zombie process. Until the parent does this `wait()`, or until the parent dies and `init` does this `wait()`, the process will remain in the zombie state until reboot.

20. What does it mean to increase a processes nice value? What happens to the process?

Increasing the nice value of a process causes the process to receive less CPU time by negatively influencing its priority. Thus, all higher priority jobs will run ahead of, and run more often, than those processes with higher nice values.