Advanced Unix System Administration
Spring 2008
Homework 2

This assignment is due via email to <sluo+decal@ocf.berkeley.edu> by 11:59 PM on **Thursday, March 13** (I'm willing to extend this to the following Monday, if people want).

1. *POSIX ACLs, and portability considerations.* I noted in class that POSIX draft ACLs don't necessarily behave the same way on all the systems where they're implemented. While the behavior of the ACL access checking doesn't differ the way I implied it did in class, there are plenty of user-visible differences in the utilities. Here's a chance to work with ACLs a bit, and look at portability differences between Linux and Solaris (specifically, Solaris Nevada build 78, a pre-release version of Solaris 11).

   The Solaris box to use is available via SSH from `tempest` to `10.20.0.22` (use your `tempest` username and password).

   a. On a Linux box (like `tempest`), create a new file with permissions `0600`, then set ACLs on it allowing user `nobody` to read it (don't forget the mask). Look at the output of `ls -l` on the file. Repeat on Solaris. What differences, if any, do you notice?

   b. Remove the ACLs on the file, on both Linux and Solaris. What did you do differently on each system?

   c. Now create a directory with permissions `0700`. Set a default ACL on it allowing user `nobody` to read files created in this directory, and `cd` into and read subdirectories of this directory. How does the procedure for doing this differ?

   d. Create a new file and a new subdirectory in this directory, and look at the ACLs that they inherit. Are they the same on both Linux and Solaris? How does the mask interact with the ACL entry for user `nobody`?

   e. Use `chmod` to add group execute permissions to the file you created in part (d). Look at the ACLs now; why did they change in this way? Compare the ACLs on Linux and Solaris. Are they the same?

   f. Set `umask 777`, then try creating a new file. Is the `umask` honored?

   g. Trace the `getfacl` command on both systems (on Solaris, you want `truss(1)`). Identify the system call used to access the ACL list. Is it the same on both systems? *Optional:* What does this reveal about the way POSIX draft ACLs are implemented on the two systems?

These sorts of portability differences drive everyone who has to work on multiple systems crazy; this is why, where possible, you want to demand that your software vendors work towards creating cross-platform standards, and adhere to existing standards to the greatest degree possible.

2. *A login session.* This problem and the next one require root access to a Unix system. You each have a virtual machine set up, accessible from `tempest` on `10.20.1.x1`, where x is the last digit of your UID on `tempest` (`id` will show your UID). Log in with your `tempest` username and password (direct root SSH logins are disabled); the root password will be the same as your password.

   a. Trace an SSH login. What UID does the login process initially run as? Must this be the case, and why? Identify the parts of the output that show when the process changes user and group IDs, and when your login shell is invoked.

   b. Look up what a POSIX "session" is, and what a "session leader" is. Give two reasons why it's important that a new session be created for a user login. (Hint: one has to do with an important feature people expect from their shells; another has to do with what happens if, say, the SSH daemon dies or is killed.) Identify in the trace output where the session corresponding to your login shell is created. Identify which process ends up being the session leader for the login session once the login shell is invoked. (If you're having trouble finding these definitions, try looking in the Single Unix Specification, available from the Open Group. I encourage you to look for other resources first, though, since, like all standards documents, the writing is extremely dense.)

   c. Look up what a "controlling terminal" is. Why is it important that a user login has a controlling terminal? Identify in the trace output when the login process acquires your controlling terminal, and give the name of the device file which corresponds to it.

   d. Outline the steps needed for a running process (such as `sshd` or `login`) to create a normal login session for a user. What is the latest point at which you could effectively set up resource limits for a login session, and why?

3. *User names and UIDs.* As I mentioned in class, the relationship between user names and UIDs is not as absolute as you might believe.

   a. Create a new user, and observe what `ls` shows the owner of that user's home directory to be.

   b. Change the username (edit `/etc/passwd` and `/etc/shadow`). Does the owner displayed by `ls` for the user's home directory change as well?

   c. Remove the `/etc/passwd` entry for your new user (make a backup first, you'll need it for the rest of the problem). Now what does `ls` show for the ownership of the user's home directory?

d. Based on the above, how do you think the ownership information is stored on disk? How do you think `ls` decides what username to display?

e. Restore the entry you removed, and create a second entry in `/etc/passwd` with the same UID, but a different username. Copy the `/etc/shadow` entry for the first user and change the username to match your duplicate user. Try logging in as both. Do things look and behave the same for both? Is there a difference in what files they can read? Can they kill each other's processes?

f. Add one of the users to a group to which it doesn't already belong, and try logging in as both users again. Do they both belong to this group? Change one of the users' passwords. Do they both have the same password now?

g. Based on the above, why might you want to have two users with the same UID? How might this be abused?

4. *A simplified set of init scripts.* I mentioned in class that reading init scripts is one of the best ways to learn how an unfamiliar, inadequately documented system is put together and how to configure it. Here's a simplified set of init scripts that does what it takes to configure and boot the system, while being easier to read through than real scripts (which have to deal with many varied configurations, errors, etc.). Note that, to avoid complexity, the configuration looks different than most traditional Unix systems.

   From `tempest`, log in to `10.20.0.21` using your `tempest` username and password.

   a. Examine `/etc/inittab`. Do these init scripts implement a SysV- or BSD-style init? How do you know?

   b. Describe in general terms the tasks that will be performed on every system boot, no matter which runlevel is selected.

   c. How would you configure a new filesystem (`/dev/hdc1`, type `ext3`, mounted on `/export` with options `nodev` and `nosuid`) to be mounted on boot? What if you didn't want it to be mounted on boot? How would you change the mount options for the root filesystem? How would you force the system to check all filesystems on the next boot?

   d. How would you add a new network interface (device name `eth1`, with IP address `10.20.42.42`, netmask `255.255.255.0`)? How would you change `eth0`'s IP address? How would you change the default route?

   e. What is the default runlevel? Describe in general terms the tasks that are performed when booting into this runlevel.

   f. Describe in general terms what tasks are performed when the system is shut down. (Hint: which runlevel corresponds to shutdown?)

   g. Suppose you wanted to have a webserver (init script `/etc/init.d/httpd`) be started on system startup and stopped gracefully on system shutdown. How would you set this up?