CS5950 Programming Assignment #2 - `shash`
Due Date: *Sunday, March 29, 2015 @ 11:59pm*

Overview

In this project, you will write a shell that restricts user command execution to a set of binaries that has been specified via an SHA1 sum. Additionally, all commands that a user executes from within the shell will be logged for audit by the system administrator. This shell might be used for guest accounts on a UNIX system, so that access could be granted, but restricted, for unknown/untrusted users.

Command execution is restricted based on the content of a configuration file. The shell reads this file on startup. The file specifies: (1) the (fully qualified) name of all binaries that can be executed by the shell, (2) the SHA1 sum the binary must have, and (3) the environment in which each binary must be executed. As commands are executed, they are logged to a specified file for inspection by the system administrator.

Configuration File

The shell will only execute commands found in the file `.shash.config` in the from which `shash` is invoked. This file contains data in a prescribed format as follows.

1. There are no blank lines.
2. The file contains one or more environment descriptions. An environment description in turn contains an environment specification (the set of strings that comprise a process' environment) followed by a command specification.
3. An environment specification is one or more (non-empty) lines, each containing one element of the environment under which the following binaries must execute. Each line contains a single variable definition followed by a newline (or EOF on the last line of the file) and has no preceding or following whitespace. The string "EMPTY" is a sentinel value that indicates the environment should be empty.
4. A command specification contains one or more lines describing the binaries that will execute under the preceding environment. Each line adheres to the following format.

   The line contains a sentinel character, a name, and an SHA1 sum for a binary file. The line must contain as its first character the symbol '*' followed by variable number of spaces or tabs, followed by the fully qualified name of a binary file, followed by a variable number of spaces or tabs, followed by an SHA1 sum, followed by a variable number of spaces or tabs, followed by a newline (or EOF on the last line of the file.)

The file `.shash.config` should not be writable by group or world. If it is, the shell should silently (with respect to `stdout`) exit.

Shell Operation

On invocation, the environment for `shash` is emptied and the configuration file is read. A log entry is created if there is an error processing the configuration file. Log the real and effective UIDs, GIDs, time, date, controlling terminal, and problem with the configuration file to `.shashLog` in the current directory.

A user may then invoke commands using **either** the fully qualified path name or the unqualified command name. Prior to executing the user's command(s) the shell will:

1. Determine that the named command is in the set of allowed commands (via its existence in the configuration file). If a fully qualified name is not entered by the user, then the first matching command name found in the configuration file will be used. (For example, if a user enters `ls` and the configuration file (first) contains an entry for `/usr/bin/ls`, then `/usr/bin/ls` will be used.) If the command is not contained in the configuration file, then the shell silently refuses to execute the command.

2. Ensure that an SHA1 sum run over the binary file matches the sum contained in the configuration file. If the sums do not match, the shell silently refuses to execute the command.
3. Create the command environment using the definition contained in the configuration file for the selected command(s).
4. Log the real and effective UIDs, GIDs, time, date, controlling terminal, command entered by the user, and created environment(s) to `.shashLog` in the current directory.
5. Execute the command using `execve`, passing in the created environment.
6. Log an error return (if there is one) from `execve` with the error return code and the time, date, and command logged prior to the execution.

A silent exit should include the message "Silent Exit" being emitted to `stdout`. The shell must handle a single pipe. For example, it must handle the command:

```
ls -lt | grep root
```

but not the command:

```
ls -lt | grep root | wc .
```

The commands `q` and `quit` should cause your shell to terminate.

Secure Coding

You must follow the secure programming guidelines covered in class. If you add significant security features that are not part of this specification, be sure to create and submit a README describing these features.

Submission

The project may be performed in groups of two. Individual work is also allowed, but no changes will be made in the project scope or submission deadline if this option is freely chosen. The following rules apply to collaboration between groups. (Of course there are no restrictions on collaboration among group members.)

1. You may discuss the program with others.
2. You cannot bring any printed/written material into the discussion with you. (You may not show anyone your code; you may not view the code of anyone else. This includes others both enrolled in the course and others not enrolled in the course.)
3. You may not generate any printed material during the discussion.
4. You may generate written material during the discussion, but it must be destroyed immediately afterward. Don't carry anything away from the discussion.
5. If you are in doubt about the acceptability of any collaboration activity, I expect you to check with me before engaging in the activity.

On the Canvas assignment page you will find example code for cleaning the environment (`cleanEnv.c`), executing a simple command pipeline (`simpleCmdPipe.c`), and computing an SHA1 sum (`EVP.c`). *Do not assume that the code has been secured.* It is expected that you have sufficient background that the code is self-explanatory. If not, see me.

Submissions will be contained in a file named `shash.tgz` and will be submitted via Canvas. Include a `Makefile` so that when I type `make` in the directory containing your recovered submission, a binary file named `shash` is created. Typing `make clean` should remove all object files and the created binary `shash`. You may use either C or C++. The projects will be graded on a Linux machine.