David Dantowitz
david@dantowitz.com
201-532-3053

Problem Description – from an actual client project

For an arbitrary set of users, commands, and objects, create a compact representation of permissions to define which users can perform which commands on which objects. Write code to parse the permissions file and authorize actions by users to objects based on those permissions.

Your solution should enable a system admin fine and course grained control to indicate whether one, N, or all users, commands, and objects are permitted combinations. The admin need not specify all users, commands, or objects in the permissions file, thus your system will have the ability to use defaults and use wild cards for information not explicitly defined.

Creating a flexible representation is as important as the code.

1) Define how a system admin will represent permissions.
2) Write code to read the permissions file and store in RAM.
3) Write code to determine if a *command* issued by a *user* on an *object* is permitted. When called, your function should return 1 if the action is permitted and 0 if not permitted, e.g.:

   authorized = isAuthorized(user, command, object);

Users, commands, and objects are strings.

Examples and Hints to Think About

An admin may wish to disable all access to all objects. Or, enable all commands by all users, except, let's say "delete" which can only be used by a particular group of users. There's another concept, enable the admin to specify a group of users and assign them permissions.

You may consider defining shorthand for the group of *all users*, perhaps "*", so an admin could specify:

user:* **may not execute** the **delete** command

If you wanted to override the "delete ban" for a specific user, enabling *only* them to delete, you might use:

user:**userA may execute** the **delete** command

As part of your solution you'll have to deal with precedence. If you prefer an order dependent text file, please describe why. For now, if you have the two declarations above in your permissions text file, the result of calling **isAuthorized** would be independent of the order of the two lines (that is, a specific user directive overrides a larger, group directive). We can assume, for this exercise, that you have an intelligent admin or a validation tool to handle conflicts, thus, the input the file will be considered to be unambiguous in terms of its declarations.

In the case above we see the hint that precedence is an important part of the solution. It makes sense that declarations for *all users* are overridden by declarations for groups of users and those would be overridden by declarations for specific users.

Just as "*" may be used to indicate *all users*, it is helpful to indicate a shorthand for "every command." For example:

user:**super may execute any** command

Finally, it may be helpful to indicate control over a specific object, e.g., a demo object that would permit all commands to be used by a demo account.

user:**demo may execute any** command on object **demoFile**

This last directive has an example of a single, specific object. In previous example directives "all objects" have been implied by default. In this one, we limit the directive to a single object.

You may observe that these examples are a bit wordy, it's more likely a line in your permissions file would be more compact, for example:

any `-*`

Which would indicate, "for any command, all users are not permitted" (e.g., unless overridden, no users could execute any commands). This would be the default state of an explicitly closed system. You would then add directives to indicate which users-command-object combinations would be permitted.

Note that these are only a few examples and shouldn't restrict your design ideas.