CS 465: Introduction to Computer Security
Programming Assignment #2

Assigned: Thursday, Nov 5, 2015
Deadline for submitting the assignments electronically: 11:59 pm Wednesday, Dec 2, 2015
Deadline for the hard copy: at the beginning of the class Thursday, Dec 3, 2015
No late submissions will be accepted.

The goal of this assignment is to implement a system which emulates a simplified version of the access control mechanisms used by the Windows XP operating system. The system consists of user accounts, groups of users, and files. There is a default privileged user, called `administrator`, who is able to perform administrative tasks such as creating user accounts and groups, and assigning permissions. Each user account has a username and password. A regular user may belong to zero or more groups. Each file has an owner and various access permissions associated with the owner and optionally with other users and groups. As with any trusted operating system, your system will log its activity in an audit log file.

The programming assignment may be implemented in any programming language available on the LOUD environment or LCSEE shell servers. The program must be able to be compiled and run on these environments.

The program executable must be named `access`, or the closest possible equivalent depending on the programming language that you chose. For example if you use Java, the file name for the class containing the main method should be `access.class`. The program will be invoked by the following command line (or equivalent depending on programming language),

```
./access filename
```

where `filename` is a command line argument that stands for the name of an ASCII file containing the instructions that the program should interpret. An example of this could be:

```
./access testcase1.txt
```

The program must accept the name of a text file as input via a command line argument and respond appropriately based on a predetermined set of instructions in the file. Each instruction will be written on a separate line. It can be *assumed* that all instructions will be free from any syntax errors, that is, each line in the input file will contain a syntactically correct instruction. This will cut down on time spent implementing typical error handling.

Here is a list of commands with options that should be implemented:

- **login** *username password*
- **logout**
- **net user** *username password*
- **net group** *groupname* {*username*}
- **create** *filename*
- **read** *filename*
- **write** *filename text*
- **execute** *filename*
- **xcacls** *filename* [options]
- **end**

In order to create the default privileged user `admin`, the default `Administrators` and `Users` groups, the **first four lines** of the instructions file has to be of the form:

```
net user admin password            (creates the user admin)
net group Administrators           (creates the group Administrators)
net group Administrators admin     (adds admin to Administrators group)
net group Users                    (creates the group Users)
```

This will be the ONLY time when these commands can be executed without a user being logged in or by a user other than the `admin` (or who is a member of the `Administrators` group). If the very first command is not of the form described above, then your program should report an appropriate error (both to the console and audit log) and terminate. After execution of these commands, the `admin` user will still need to login to perform any further actions.

A username, group name, or filename may consist of up to 30 ASCII characters except for a forward slash ('/'), colon (':'), or any of the following white space characters: carriage return, form feed, horizontal tab, new line, vertical tab, and space. A password may consist of up to 30 ASCII characters except for the following white space characters: carriage return, form feed, horizontal tab, new line, vertical tab, and space. Instructions, usernames, group names, and filenames are **not** case sensitive however passwords are always case sensitive.

Each user account has a password. A list of users and corresponding passwords should be maintained in an ASCII text file called *accounts.txt*. You may organize this file any way that you like as long as it is used by your program for user authentication AND you clearly and thoroughly explain your approach and the format of your *accounts.txt* file in your program documentation.
By default all users are members of the group called `Users`. A user can be a member of multiple groups. Your program should use an in-memory data structure to keep track of user groups. After all instructions have been processed, but prior to terminating, your program must create a file named `groups.txt`. The `groups.txt` file should contain a listing of each group and the users who belong to the group. The exact file structure will be provided as part of example test cases.

Every file on the system has associated with it an owner and an Access Control List (ACL). The ACL contains access control entries (ACE) for each security assignment. There is an ACE for each of the following: the file owner, other users, and groups. Your program should use an in-memory data structure to keep track of this file meta-data during execution. After all instructions have been processed, but prior to terminating, your program must create a file named `files.txt`. The `files.txt` file should contain a listing of each file name and owner along with its associated ACL. The exact file structure will be provided as part of example test cases. For each file created with the command `create`, your program must also create a physical text file and provide the ability to read from and write to the file. The file should be created in the same directory as the program executable (i.e. the "current" directory).

Upon the execution of each instruction, the program should DISPLAY an appropriate comment on the screen AND LOG the SAME comment into an audit log file called *audit.txt*.

Since the *accounts.txt, audit.txt*, *groups.txt*, and *files.txt* files serve a special purpose (system files) they should not be accessible to any user via the `create`, `read`, `write`, `execute`, or `xcacls` commands. An appropriate error message should be displayed and logged if any of these file names are referenced using any command.

The user accounts, groups, files, and permissions should not persist across multiple executions of your program. That is, each execution of your program should start with a "clean slate". (The *accounts.txt, audit.txt*, *groups.txt*, and *files.txt* files should be completely overwritten if they already exist when your program first starts. This is also the case with any files that were created via the `create` command. If a file already exists when your program starts, its content should be overwritten. However, subsequent calls to the `write` command that are issued during a single execution of your program should append to the appropriate file.)

Detailed description of commands

`login` *`username password`*

- Example: `login name cs465rules`
- Checks username and password against records in the *accounts.txt* file. If the username and password are correct, logs the user into the system. Otherwise, displays and logs an error message, then denies login.
- Things to consider / constraints:
    - If a user is currently logged in, another user cannot login. (This system does not support concurrent users.)
    - Once logged in, the user will be able to use `create`, `read`, `write`, and `execute, and xcacls, (and net user and net group if an Administrator)` instructions which are allowed or denied according to the NTFS file access rights security model.
    - The user will also be able to use the `logout` as specified later in this document.
    - There MUST be a user logged in to be able to use any command other than `login`.

`logout`

- Example: `logout`
- Logs out the currently logged user

`net user` *`username password`*

- Example: `net user name cs465rules`
- Creates a user with a specified username and password
- Username and password are stored in an ASCII file called *accounts.txt* for later use (Your program has to reference this file for user authentication when executing the login instruction).
- Things to consider / constraints:
    - All users should be automatically added to the `Users` group by this function during account creation.
    - Two users with the same user name MAY NOT be created
    - This instruction may only be executed by a member of the `Administrators` group. (The only exception is the first instruction which creates the `admin` account, as previously discussed.)

net group *groupname*

- Example: `net group students`
- Creates a group to which users may be added.
- Your program should use a data structure to maintain a listing of all groups that have been created and the user(s) that belong to each group.
- Things to consider / constraints:
    - This command can only be executed by a member of the `Administrators` group.
    - A group is initially created without any users belonging to it.
    - Two groups with the same name MAY NOT be created

net group *groupname username*

- Example: `net group students name`
- Adds the specified user to the specified group
- Things to consider / constraints:
    - A user can be a member of multiple groups
    - If the specified user or group does not exist, then an appropriate error message should be displayed and logged.
    - This instruction may only be executed by a member of the `Administrators` group

xcacls *filename* [options]

This function is used to change the access permissions (ACL) associated with a file from a command-line. This is a simplified version of the Extended Change Access Control Lists tool utility that is included with the Windows XP Resource Kit available on the original OS disc or downloaded from Microsoft. The OS as installed by default has the standard cacls.exe available. Windows Vista and Windows 7 have a newer version called icacls.exe with even more functionality.

- ***filename*** indicates the name of the file to which the ACL or access control entry (ACE) is applied.
- **Options** include /G `user|group:perm`, /P `user|group:perm`, `/D user|group`, and `/E. .`

- **`/G user|group:perm`** This option grants a user or a group access to the file as follows. The *perm* (permission) variable applies the specified access right to files. The *perm* variable accepts the following values:
    - R Read
    - W Write
    - X Execute
    - F Full control (all of the above)

  For example, only the user `name` will be granted full control access to the `Test.dat` file if you run
  `xcacls Test.dat /G name:F` command. All ACEs applied earlier are lost, including the default permissions!

- **`/P user|group:perm`** This option replaces the existing access rights for the `user or group specified`. The values for specifying `perm` are the same as for the `/G` option.

- **`/D`** ***`user|group`*** This option explicitly denies all access for a specified user or group. Explicit deny option takes precedence over any allow permissions in place. For instance, assume a user is a member of a group that has been given access to a file, but then the user has explicitly been denied the access to that file. This user will not have access to the file. Another example, if the user is a member of two groups, one with access to a file, and another with explicitly denied access to the same file, that user will be denied access. The /E option must be used with /D or else the existing ACL will be overwritten including all default permissions. For example if the current ACL for *file.txt* is {admin:F, students:RW, Bob:X} and you issued xcacls file.txt /D fred, then the new ACL would be {fred:D}

- **`/E`** This option can be used in conjunction with the /G, /D, and /P options to edit the ACL instead of replacing it. The new user|group:perm (ACE) should be appended to the end of the ACL.

- Things to consider / constraints:
    - A user must be logged in to execute this command
    - This command can be executed only by the file owner or an Administrator otherwise an error message should be displayed and logged.
    - **The /E option must be used with /G,/D, or /P to not overwrite existing ACL.**
    - If the specified file does not exist then an appropriate error message should be displayed and logged.

`create` ***`filename`***

Example: `create newfile.txt`

- Creates an empty file with default permissions.
- The default permissions for a created file are as follows:
    - The file owner/creator and the `Administrators` group: **F**ull control. These permissions can later be changed, removed, or overwritten by the xcacls command. (The file owner cannot be changed. This functionality is not included in the assignment.)
    - Additionally, if the file was created by a member of the `Administrators` group then the `Users` group is assigned **R**ead access as well.
    - No other users or groups are allowed any access.
- Your program should use a data structure to keep track of the file name, owner, and ACL.
- Things to consider / constraints:
    - If the file already exists, then an appropriate error message should be displayed and logged.
    - A user must be logged in to execute this command
    - The owner of the file should be the user who is currently logged into the system and issued the `create` command

`read` *`filename`*

Example: `read file.txt`

- Displays the contents of a file
- Access to this command is granted or denied according to the NTFS file access rights security model as described in this assignment.
- The current user is denied read access if an explicit denial exists for the user or a group the user belongs to in the ACL, otherwise:
    - If the current user is the owner of the file and read or full control access for owner has been granted, then the instruction may be carried out.
    - Else if the current user is not the owner, but is another user or a member of a group that have a read or full control access to the file then the instruction may be carried out.
    - Else if the current user is not the owner, is not another user or a member of a group that has been granted read or full control access or to the file, but read or full control access has been granted for the `Users` group, then the instruction may be carried out.
    - Otherwise access should be denied.
- Things to consider / constraints:
    - If the file does not exist, then an appropriate error message should be displayed and logged.
    - A user must be logged in to execute this command

`write` *`filename text`*

Example: `write file.txt some text to put in file`

- Appends a line of text to a file.
- Access to this command is granted or denied according to the NTFS file access rights security model as described in this assignment.
- The current user is denied write access if an explicit denial exists for the user or a group the user belongs to in the ACL, otherwise:
    - If the current user is the owner of the file and write access for owner has been granted, then the instruction may be carried out.
    - Else if the current user is not the owner, but is another user or a member of a group that has a write access to the file then the instruction may be carried out.
    - Else if the current user is not the owner, is not another user or a member of a group that has been granted write access to the file, but write access has been granted for the `Users` group, then the instruction may be carried out.
    - Otherwise access should be denied.
- Things to consider / constraints:
    - If the file does not exist, then an appropriate error message should be displayed and logged.
    - A user must be logged in to execute this command
    - The text should be APPENDED as a new line to the end of the file. Make sure your program does not totally overwrite the file.

execute *filename*

Example: `execute prog`

- Emulates the execution of a file
- This should determine whether or not execution is permitted, then display an appropriate message such as "*prog executed successfully*" or "`User is denied execute access to prog`"
- Access to this command is granted or denied according to the NTFS file access rights security model as described in this assignment.
- The current user is denied execute access if an explicit denial exists for the user or a group the user belongs to in the ACL, otherwise:
    - If the current user is the owner of the file and execute or full access for owner has been granted, then the instruction may be carried out.
    - Else if the current user is not the owner, but is another user or a member of a group that execute or full access to the file has been granted then the instruction may be carried out.
    - Else if the current user is not the owner, is not another user or a member of a group that has been granted execute or full access to the file, but execute or full access has been granted for the `Users` group, then the instruction may be carried out.
    - Otherwise access should be denied.
- Things to consider / constraints:
    - If the file does not exist, then an appropriate error message should be displayed and logged.
    - A user must be logged in to execute this command

end

Example: `end`

- The last instruction in the input file.
- Indicates that your program should write out the *groups.txt* and *files.txt* files as described previously, perform any other necessary cleanup actions (i.e., closing open file streams) and terminate.

Note: The previous commands (`logout, net, and xcacls`) are scaled-down versions of the actual commands available in Windows XP. Some optional parameters and permissions have been omitted or altered slightly for simplicity. Also, in the real XP environment, any user can run '`net [options]`' or '`xcacls` *filename'* to view existing users, groups, and permissions.

Extra credit (Up to 20 points)

The goal is to implement a simplified version of the Windows 7 User Account Control (UAC). There are three settings of the level of UAC:

- Always notify: As in Windows Vista UAC will always prompt the currently logged in user for permission.
- Notify only on change: UAC prompts the currently logged in user for permission only when a program attempts to make a change (execute, create file or write to a file). This is the default setting in Windows 7.
- Never notify: This is the least secure setting. UAC never notifies the user about any action attempted by a program.

When a user account is created, the default UAC setting "Notify only on change" is recorded in the *accounts.txt* file. The user can choose to change the default UAC setting by using the following command

`uac setting`

- Example: `uac Always`
- Sets the UAC for the currently logged in user to the specified setting by changing the record in the *accounts.txt* file. The parameter `setting` can have one of the following values:
  - `Always:` for Always notify
  - `Change:` for Notify only on change
  - `Never:` for Never notify
- Things to consider / constraints:
  - There must be a user logged in for this command.
  - It sets the UAC only for the currently logged in user. The UACs of other users are not affected.

The actions of a program attempting to run on a user machine will be emulated by the following commands:

```
Program execute filename
Program create filename
Program read filename
Program write filename text
```

('`Program`' in above commands means that there is a program (not the user who is currently logged in) attempting to execute, or create, read or write to a file.

Your program will check the UAC setting for the currently logged in user (which is written in the *accounts.txt* file). The action will depend on the setting.

- If the setting is `Always` the user will be prompted for permission for any action requested by the `Program.`
- If the setting is `Change` the user will be prompted if the `Program` attempts to make a change (that is, `execute`, `create` file, or `write` to a file). If the `Program` attempts to read from a file the currently logged in user will not be notified.
- If the setting in `Never` the user will not be notified for any action attempted by the `Program.`
- Things to consider / constraints:

- The prompt for permission should appear on the screen, after which the user should enter a response, either "`yes`" or "`no`", on the command line. Granting or denying access will depend on the user answer ("yes" or "no") and the access of the user to that particular file as explained next.
- Note that the `Program` can have a particular access to a file only if the currently logged in user has the desired access to that file. For example, the `Program` may request read access to a file homework1.docx, but if the currently logged in user does not have read access to that file the `Program` will be denied access regardless of the UAC settings of the user.
- The commands run by a `Program`, assuming the access is granted, have the same effect as the corresponding commands run by the user (e.g., will add text to a file, log the action in the *audit.txt*, etc.)
- If the file does not exist, then an appropriate error message should be displayed and logged.
- A user must be logged for these commands to be used.

If you chose to implement the extra credit functionality, your program should NOT require any modifications to the syntax for ANY of the other instructions. Additionally, if you include the extra credit functionality, you **must** state so in your documentation status sheet.

Notes:

Do not forget to turn in the documentation, which should include at least two test cases you used to test your program, each consisting of a sample input file and the corresponding output provided by your program. These test cases should be different from the ones provided on the class Web page.

Like all assignments in this class you are prohibited from copying any content from the Internet or sharing ideas, code, configuration, text, or anything else, or getting help from anyone in or outside of the class. Consulting online sources is acceptable, but under no circumstances should anything be copied. Failure to abide by this requirement will result in zero points for the whole programming assignment.