2019

Project: Password Management System

COMP30640 – OPERATING SYSTEMS

STUDENT: THOMAS GROGAN STUDENT NO: 18203315

Introduction:

This project involves the creation of password management system using the Bash command language. It is built on a client-server model.

Requirements:

- The system should store login ids and passwords for various users for their different services. An example of these services could be a bank or a social media account.
- A number of bash scripts were written to implement this system
- Firstly, the init.sh script is used to create a new user.
- Secondly, the insert.sh script is used to create a service with its login details and password within the specified user's directory. This script includes the option to update the login and password for a service by including 'f' as the third parameter.
- Thirdly, the show.sh script is used to display the login details for a service of a given user.
- Fourthly, the rm.sh script is used to remove all information relating to a specified service from the system.
- Finally, the ls.sh script is used to display the services stored by a specified user and/or folder.
- The system also consists of server.sh and client.sh scripts. The server.sh script, naturally, acts as the server for the system while the client.sh script provides the user interface

Architecture:

- This init.sh script checks to see that only one parameter is given. If not, it prints an error message. An error message is also printed if given username already exists in the system. Otherwise, it creates a directory using the username provided and prints a message to that effect.
- The insert.sh script checks that four parameters were given. If not, it prints an error message. An error message is also printed if the first parameter, the username, does

not already exist on the system. If the third parameter entered is ‘f’, then the script continues by updating the login details of the service file or, if the service was not already created, creating a new service with those login details. If the third parameter is not ‘f’ the system prints an error message or, if the service was not previously created, creates a new service file with the given login details.

- The show.sh script checks that two parameters were given and, if not, prints and error message. If the username, first parameter, or service, second parameter, do not exist error messages are printed. If all conditions are satisfied the script displays the contents of the specified service’s file.
- The rm.sh script checks that two parameters were given and, if not, it prints an error message. If the username, first parameter, or service, second parameter, do not exist error messages are printed. If all conditions are satisfied the specified service file is deleted from the system.
- If the number of parameters is not equal to one or two the ls.sh script prints an error message. If only one parameter is given the script checks if that user exists and uses the tree command to display the services in that user’s directory. If the user does not exist an error message is printed. If two parameters are given the script checks if the user and folder exist and uses the tree command to display the services within the specified user’s folder. If either the user or the folder do not exist, error messages are printed.
- The server.sh script creates a pipe called server.pipe when it is run, if the pipe does not already exist. It then reads from server.pipe and assigns items from the array to variables called client, request, user, service and details. While a lock for the given user exists the the execution sleeps for 1 second. If the user.lock does not exist a directory with the given username, user.lock, is created. A case statement then follows. Depending on the request variable, one of the scripts described earlier is called with the appropriate parameters. The output of the script’s execution is redirected to the client pipe, client.pipe, and the lock directory, user.lock, is deleted. If the request is ‘shutdown’ a message to that effect is sent to the client pipe, the lock directory is deleted, and the server pipe is also deleted. If the request is not in the case requests an error message is printed and the lock directory is deleted.

- The client.sh script prints an error message if the number of parameters is less than two. Otherwise, it checks to see if the second parameter is equal to init, insert, show, ls, edit, rm, or shutdown. If it is init, the script checks that the number of parameters is equal to three. It then creates a pipe using the first argument, the client id. It sends the three arguments to the server pipe, reads the response from the client pipe, prints the response and then removes the client pipe. If it is insert, the script checks that four parameters were given. It then creates a pipe using the client id. It asks the user to enter a login and asks the user if they would like to use an automatically generated password. If the user enters 'Y' or 'y' then the script automatically generates a password. If the user enters anything else they are prompted to enter a password. The login and password are assigned to the $payload variable and sent along with the other four parameters to the server pipe. The response is read from the client pipe and printed. The client pipe is removed. If the second parameter is edit, the script checks that four parameters were given. It then creates a pipe using the client id. It sends parameters one, three, four and 'show' to the server pipe and assigns a temporary file to the $temp variable. It reads the $payload from the client pipe and sends that $payload to the temporary file. The temporary file is opened to edit using vi. The scrip reads from the temporary file and sends this as $payload along with parameter one, 'update', parameter three and parameter four to the server pipe. The response is read from the client pipe and printed. The temporary file and client pipe are both removed. If the second parameter is rm, the script checks that four parameters were given. It then creates the client pipe and sends the four parameters to the server pipe. The response is read from the client pipe and printed. The client pipe is removed. Finally, if the second parameter is shutdown, the script creates a client pipe, sends the parameters to the server pipe, reads the response from the client pipe, prints the response and removes the client pipe.

Challenges:

- The first significant problem I encountered was with the insert.sh script. It occurred when the second parameter was included a forward slash, eg. 'Bank/aib'. I needed to separate the two parts of the parameter into a directory and a file. To do this I changed

the IFS (Internal Field Separator) to '/' if the parameter contained a forward slash. The parameter is then read into an array and the first item in the array I assigned to the folder variable and second item is assigned to the file variable. This then allows for the creation of a directory and file using the two variables, respectively.

- The problem of concurrency arose during the server section of the project. I decided to use a locking mechanism which in which a lock is created using the username provided as the second argument from the client script. The lock is in the form of a directory as 'mkdir' is an atomic process. While the lock is in existence it prevents clients from accessing the specified user's folder. It is removed after the request from the client has finished its execution.
- I had some trouble with the 'show' request in the client.sh script. I decided to read the response from the client pipe into an array and then assign array item one and three to the variables $login and $password which would be used in the echo statements.
- I experienced some issues with the 'edit' section in the client.sh script. This involved the creation of a temporary file which would hold the current login details which could be edited in order to update them. I overcame these issues by researching about the 'mktemp' command and found a way to assign the command to a variable which I called $temp. I used vi to allow the user to edit the contents of the temporary file.
- I decided to give the user the option to automatically generate a password when entering a new service as part of the bonus section. The method I have used to generate a password is 'date +%s | sha256sum | base64 | head –c 32'. It uses SHA to hash the date, goes through base64 and outputs the top 32 characters. I found this method on the howtogeek.com website.

Conclusion:

This was a very interesting and worthwhile project, which I learned a lot from. It has been great to be able to put our work from the lectures and practicals into practice. There may be some shortcomings in my solutions. However, I hope that I have satisfied the requirements that were set out in the brief for this assignment.