School of Computing & Information Technology

CSCI262 System Security
Spring 2024

Assignment 1 (10 marks, worth 10%)
Due date: 30th August 2024, 23:55 (Sydney Time).

Part One: Short answer questions: 4 Marks

1. Determine the entropy associated with the following method of generating a password. **1 Mark**

```
Choose, and place in this order, one lower case letter, following by one upper
case letter, followed by two digits, followed by @, followed by two letters,
each upper or lower case, and then followed by four symbols drawn from the set
{$,7,3,v,w,J,z,T}.  Finally, apply the hash function Tiger to give an output
string in hex which will be used as a password.
```

Assume random choices are made with equal likelihood of each symbol from the space being chosen from. So for a random digit there are 10 possibilities, each chosen with probability 1/10.

2. For the following collection of statements, describe the sets of actions, objects, and subjects; and draw an access control matrix to represent the scenario. **1 Mark**

```
Alice can climb trees and eat apples.

Bob can climb fences, eat apples, and wave flags.

Trees can hurt apples.

Carol can jump waves, eat apples, and wave flags.
```

3. Assume an application requires access control policies based on the applicant's age and the type of funding to be provided. Using an ABAC (attribute-based access control) approach, write policy rules for each of the following scenarios:

(a) If the applicant's age is more than 35, only "Research Grants (RG)" can be provided. **1 Mark**

(b) If the applicant's age is less than or equal to 35, both "RG and Travel Grants (TG)" can be provided. **1 Mark**

Part Two: Authentication and access control system 6 Marks

You are to implement a simple "file system" with login authentication and access control. Specifically:

- Construct a hash/salt/shadow based user/password creation system.
- Construct a hash/salt/shadow based user authentication system.
- Construct an associated file system, into which a user can log. Files can be created, read from, written to, but only in accordance with a four–level access control model.
- The levels of the four–level access control model are 0, 1, 2 and 3. 0 is dominated by 1, 2 and 3; and 1 and 2 are dominated by 3; and 1 is dominated by 2.

Remark: You do not need to have an actual file system, simply an internal collection of records at the levels specified.

You can implement the program in C++, C, Python or Java. You will use MD5 in this task.

The initialisation details 2 Marks

Your program will, initially, need blank files `salt.txt` and `shadow.txt`.
Running your `FileSystem` with the instruction

```
FileSystem -i
```

runs the hash/salt/shadow based user/password creation system. This program should prompt for a username, something like...

```
Username: Bob
```

Check if the username exists already. If it does, terminate the program with an appropriate notification to the user. If it doesn't, request a password with something like ...

```
Password: ........
Confirm Password: ........
```

You should add some appropriate checks on the password and give a warning if the user fails to meet a requirement. The warning should explain what the requirements are. Assuming the passwords are acceptable and the same, we make a final request of the user, something like ...

```
User clearance (0 or 1 or 2 or 3): 1
```

Once we have this information we can modify the `salt.txt` and `shadow.txt` files to include this user. To `salt.txt` we add a line, with a generic example and a specific one for user Bob given here:

```
Username:Salt
Bob:38475722
```

where `Salt` is a randomly chosen string of 8 digits.

We also add a line to `shadow.txt`, with a generic example and a specific one for user Bob given here:

```
Username:PassSaltHash:SecurityClearance
Bob:dd2da44f4437d529a80809932cb3da83:1
```

PassSaltHash is generated as the MD5 hash of the concatentation of the user's password with the salt, For example if the Password is "alphabet" and the Salt is "12345678", we would pass "alphabet12345678" to the MD5 function.

Logging in 2 Marks

Running `FileSystem` with no arguments will allow a user to try and log into the file system.
Remark: The file `Files.store` needs to be loaded, see later as to what this contains.

```
Username: Bob
Password: ........
```

The system checks if the Username is listed in the file `salt.txt`. If the Username is in the file then their salt value is retrieved and the PassSaltHash is generated. A message should be displayed to indicate that the salt has been retrieved.

```
Bob found in salt.txt
salt retrieved: Salt
hashing ...
hash value: PassSaltHash
```

The system should now compare the PassSaltHash value with that in the file `shadow.txt`. If the information in `shadow.txt` doesn't match the generated information, `FileSystem` should stop with appropriate error messages.

If the `shadow.txt` information matches, the clearance of the user is reported, and authentication is reported to be complete.

```
Authentication for user Bob complete.
The clearance for Bob is 1.
```

Once logged in ... 2 Marks

A list of allowed actions is now displayed

```
Options: (C)reate, (A)ppend, (R)ead, (W)rite, (L)ist, (S)ave or (E)xit.            (1)
```

The `C` option will result in a request for filename from the client.

```
Filename: alpha
```

The classification level of a "file" is the same as its owner's clearance level.

The program should maintain a list of "files" as internal entries. If the passed file doesn't exist, it's name, owner and classification should be added to the list. If the passed file does exist an appropriate message should be displayed and the system should re–display the menu marked (1).

The `A`, `R` and `W` choices each results in a request for a filename.

```
Filename: alpha
```

Again a check is made as to whether the file exists. If the file doesn't exist an appropriate error message should be provided and the menu (1) should be re–displayed. If the file does exist, a message informing success or failure will be displayed. Success or failure is determined by the relative clearance of the user and the classification of the file they are trying to access, in accordance with the Bell–LaPadula model. Subsequently the menu (1) should be re–displayed.

The `L` option lists all files in the FileSystem records. The `S` option saves all the data to a file `Files.store`. This file should be human readable. This file should always be loaded if it is available when `FileSystem` starts without the `-i` argument.

The `E` option should exit the `FileSystem`, after checking with the user:

```
Shut down the FileSystem? (Y)es or (N)o
```

NOTE: When your program starts, before bringing up a prompt, it should report a test output of the MD5. You should call your MD5 with the string "This is a test".

```
MD5 ("This is a test") = ce114e4501d2f4e2dcea3e17b546f339
```

Don't hard code this output.

NOTE ON SUBMISSION

1. Submission is via Moodle.
2. Include the compilation instructions with your submission in a file `readme.txt`. That file should also contain a description of how you do the reduction. If no such file exists in your submission then you will get **zero** for this Part two.
3. Make sure your program run perfectly on CAPA before submission. If it does not run on CAPA when testing then you will get **zero** for this.
4. Late submissions will be marked with a 25% deduction for each day, including days over the weekend.
5. Submissions more than three days late will not be marked, unless an extension has been granted.
6. If you need an extension apply through SOLS, if possible **before** the assignment deadline.
7. Plagiarism is treated seriously. Students involved will receive **zero**.