Finvest Holdings

Security Software & Analysis

SYSC 4810 Assignment

Nathan MacDiarmid

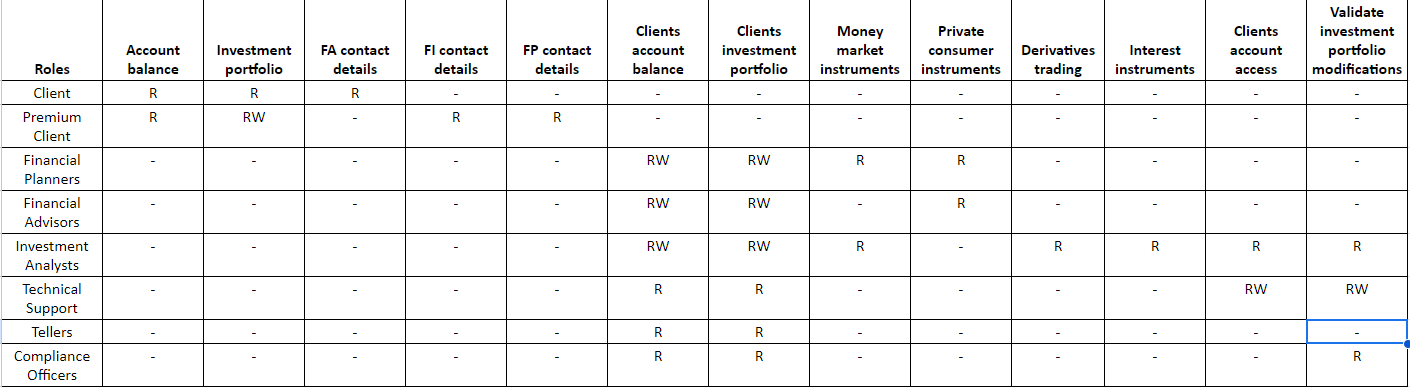
101098993

Passwords – IloveCats2!

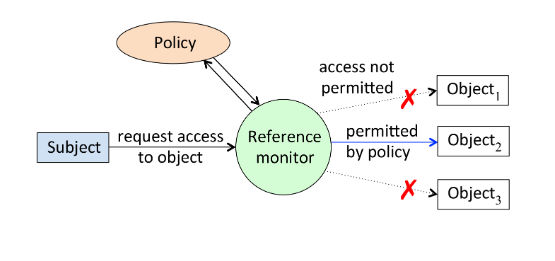
December 4th, 2023

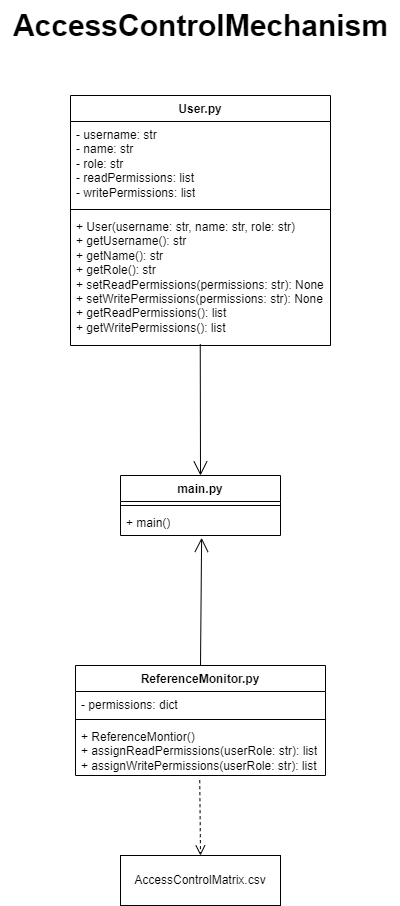
# Problem 1

1. The access control model that the Finvest Holdings security system will be the RBAC control model. This is because RBAC sets permissions based on roles which is similar to what is described in Part 1 Section 2 – Context. RBAC also allows for the expandability of permissions via additions of different roles, making it very scalable.
2. The access control representation that the Finvest Holdings security system will be is an access control matrix. This is because each of the roles have different access to various permissions and an access control matrix allows for the setting of permissions for each role separately. Furthermore, each row can be used as a capabilities list, depicting the permissions of each role, and each column can be used as an access control list, depicting how many roles have access to a selected permission.
3. In my access control matrix, a role can only have read, write, or no permissions at all. Since there didn’t seem to be any programs that required running in the description as a role would either view the information or edit that information, I chose to only allow for the options of reading and writing. Upon checking if the User is a Teller, the User will be notified the times they are allowed to access the program.

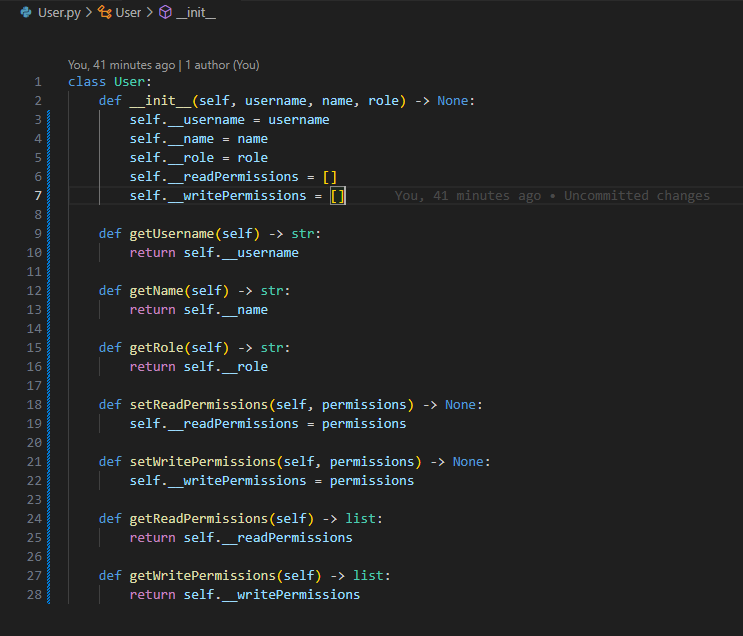


1. The access control mechanism was implemented in an object-oriented way such that it models the example of a reference model implementation that was given in class in slides Ch5\_2[3].





In the example, the Subject requests permission access from the Reference Monitor and the Reference Monitor assigns some or no permissions based on a Policy. In my access control mechanism, the User is a Subject requesting permissions from the ReferenceMonitor that is determined by the AccesesControlMatrix. Each User contains a username, a name, a role, readPermissions and writePermissions which are all private variables and can only be accessed via getter and setter methods. The username is a unique name only associated to that User and treated like a user ID. The name is the Users personal name, and the Users role is the permissions role such as Client, Premium Client, etc. The readPermissions and writePermissions contained the appropriate permissions that were sent back from the ReferenceMontior.



The Reference Monitor in the example controls what Subjects get access to what permissions based on a Policy. In my access control mechanism, the ReferenceMonitor reads from the AccessControlMatrix file and stores all the permissions in a private dictionary variable called permissions. The ReferenceMonitor has two methods that control what permissions a User receives called assignReadPermissions() and assignWritePermissions(). Like their names suggest, they assign a User their read and write permissions respectfully. They each do this by receiving a User’s role that’s passed into the method and returning a list of permissions based on that specific role. If the User role that’s passed does not exist, or has no permissions, these methods return an empty list, ensuring that User’s are only passed the proper rules.

A screen shot of a computer program

Description automatically generated

The main file is where everything is instantiated, and different User’s get their defined roles based on the ReferenceMonitor’s AccessControlMatrix. One instance of the ReferenceMonitor is instantiated as well as the list of User’s that were provided in the assignment description. These User’s are only used as an example. The

# References

1. <https://www.linkedin.com/pulse/comprehensive-overview-access-control-models-rbac-abac-jay-/>
2. <https://docs.python.org/3/library/hashlib.html>
3. https://docs.python.org/3/library/getpass.html