SYSC4810

User Authorization and Authentication System Design Specifications

Finvest Holdings

Guy Morgenshtern

[redacted]

1 The Access Control Mechanism

1.1 Access Control Model

The following combination of access control models have been chosen for this system.

Role-based Access Control (RBAC) Will clearly define roles (i.e. Client, Financial Advisor, etc..) and differentiate their permissions.

Attribute-based Access Control (ABAC) Will be used to handle special conditions on specific roles. For example, the Tellers ability to access the system depends on the time of day, an environment attribute.

This combination keeps the separation of permissions simple and fine grained enough that Finvest could add, remove, or edit groups easily while still maintaining the principle of least privilege.

Each role will have its own set of permissions attached to it, and any special conditions will be handled by attributes (i.e. *Teller* accessing the system during business hours).

This approach does include some redundancies with defining roles as *Financial Planners, Advisors, Technical Support*, etc all having the basic *Finvest Employees* permissions but with a few modifications or additions. However, these roles can be viewed as hierarchical with "*Employee*" being the base role. Additionally, in a system as small as this one, redundancy will not hurt performance and will help ensure that the permissions per role are clearly followed.

1.2 Access Control Representation

RBAC will be represented by an "Access Control Matrix" which will help define all the permissions each role will have in an easy to read, tabular way. Each row of the matrix corresponds to a role (or subject), each cell representing the permissions for a given function (or object). Since the role base of the Finvest Holdings system is relatively small, using an "Access Control Matrix" is both readable and easily maintainable.

While the matrix handles all static permissions, the purpose of the Policy Store is to handle special conditions, additional attributes, and environmental conditions as they relate to resource authorization.

In the case of the current Finvest Holdings system, the only special policy is the withdrawal of access to Tellers outside of business hours.

- *Teller*s can only access the system past business hours (9:00 am - 4:00pm).

A Policy Store in addition to the matrix adds granularity to the access control system. A policy is used to either make an exception or define additional attributes that determine access. By using a Policy Store, the addition of new policies is trivial as the store itself handles the determination of access. The system admin would only have to write the condition under which the policy applies. Additionally, granularity in the design, along with the representation of policies as objects promotes the "Modular Design" principle.

As there is only one special condition, a "Policy Store" can easily be implemented in combination with the "Access Control Matrix" without creating much additional complexity. The determination of permission will depend on both the "Policy Store" and "Access Control Matrix". First, the "Policy Store" will be queried, where the specific Teller condition will be determined using the current time and date. Then, if the policy does not apply, the matrix will be consulted. A more in depth explanation is included in 1.3.

1.3 Design and Implementation of the Access Control Mechanism

The design of this system uses the Policy Store as a determination that the *subject* **does not** have access to a given *object*. Meaning if the policy store determines that a condition is satisfied, access is denied. Otherwise, the matrix is consulted.

```
class AccessControlSystem:
   ...
   def check_authorization(self, subject, object) -> bool:
        # in this system, the access control matrix is responsible for determining who has access to the system
          # the policy store handles special conditions (i.e Tellers can't access system past business hours)
        # if the policy store evaluates TRUE on a subject-object pair, then access is DENIED, otherwise the matrix is consulted

       if self.policy_store.evaluate_all_policies(object, subject):
           return False
       else:
           return self.control_matrix.check_permission(subject['role'], object)
```

The Policy Store is made of a list of policies. Each policy takes 3 parameters upon creation.

Resource A string representation of the resource the policy for.

Attributes A dictionary of attributes the policy pertains to.

Environmental condition (optional) A function that defines any environmental conditions that should be met

For a policy to be determined to be satisfied, the policy resource must match the *object* passed in, all attributes in the policy must be present in the *subject* passed in, and if present, the environmental condition must evaluate to *True*.

```
class Policy:
    ...
    def evaluate(self, resource, attributes_of_user):

        if resource == self.resource:
            for user_attribute in attributes_of_user.keys():
                if user_attribute in self.attributes and
attributes_of_user[user_attribute] == self.attributes[user_attribute]:
                    if self.environmental_condition != None:
                        return self.environmental_condition()
                    else:
                        return True

        return False
```

To evaluate a *subject-object* pair, the Policy Store iterates through all policies in it. If any result in *True*, then the policy store returns *True*.

The matrix determines authorization given an *object* and *subject* as input. The *object* is the resource or action that is to be performed. The *subject* is the role of the user trying to gain access.

```
class AccessControlMatrix:
    ...
    def check_permission(self, subject, object):
        return self.permissions_by_object_subject_matrix[subject][object]
```

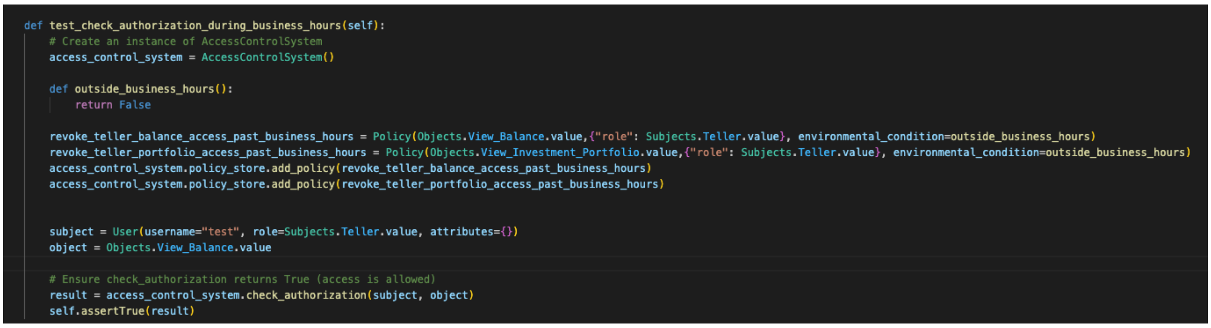
The structure of the matrix is as follows:

	View Bal	View Inv. Portfolio	Modify Inv. Portfolio	Get Contact of Fin. Advisor	Get Contact of Fin. Planner	Get Contact of Inv. Analyst	View Money Market Instrum.	View Private Cons. Instrum.	View Interest Instrum	View Der. Trading	Validate Inv. Portfolio Mod.	View Client Info	Request Client Acc. Access
Client	X	X		X									
Prem. Client	X	X	X	X	X	X							
Employee	X	X											
Fin. Planner	X	X	X				X	X					
Fin Advisor	X	X	X					X					
Inv Analyst	X	X	X				X	X	X	X			
Tech. Support												X	X
Teller	X	X											
Comp Officer	X	X									X		

1.4 Tests

In order to test the access control system properly, both the matrix and policy store should be exercised. I simulated the "outside_business_hours" function for each of the test by defining it as either true or false depending on the behaviour test

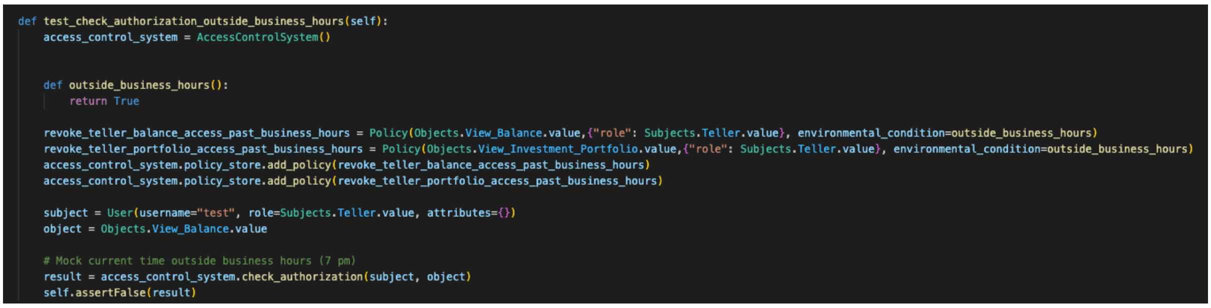
Testing the matrix:



```
def test_check_authorization_during_business_hours(self):
    # Create an instance of AccessControlSystem
    access_control_system = AccessControlSystem()

    def outside_business_hours():
        return False

    revoke_teller_balance_access_past_business_hours = Policy(Objects.View_Balance.value,{"role": Subjects.Teller.value}, environmental_condition=outside_business_hours)
    revoke_teller_portfolio_access_past_business_hours = Policy(Objects.View_Investment_Portfolio.value,{"role": Subjects.Teller.value}, environmental_condition=outside_business_hours)
    access_control_system.policy_store.add_policy(revoke_teller_balance_access_past_business_hours)
    access_control_system.policy_store.add_policy(revoke_teller_portfolio_access_past_business_hours)


    subject = User(username="test", role=Subjects.Teller.value, attributes={})
    object = Objects.View_Balance.value

    # Ensure check_authorization returns True (access is allowed)
    result = access_control_system.check_authorization(subject, object)
    self.assertTrue(result)
```

This test adds policies to the store that are checked against, but come back as false, causing the matrix to be checked. In this case, a Teller is used as it matches the policy, and the resource “View Balance” is accessed.

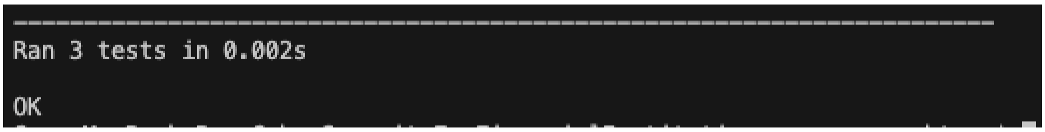
Testing the policy store:



```python
def test_check_authorization_outside_business_hours(self):
    access_control_system = AccessControlSystem()


    def outside_business_hours():
        return True

    revoke_teller_balance_access_past_business_hours = Policy(Objects.View_Balance.value,{"role": Subjects.Teller.value}, environmental_condition=outside_business_hours)
    revoke_teller_portfolio_access_past_business_hours = Policy(Objects.View_Investment_Portfolio.value,{"role": Subjects.Teller.value}, environmental_condition=outside_business_hours)
    access_control_system.policy_store.add_policy(revoke_teller_balance_access_past_business_hours)
    access_control_system.policy_store.add_policy(revoke_teller_portfolio_access_past_business_hours)

    subject = User(username="test", role=Subjects.Teller.value, attributes={})
    object = Objects.View_Balance.value

    # Mock current time outside business hours (7 pm)
    result = access_control_system.check_authorization(subject, object)
    self.assertFalse(result)
```

A similar method is used for testing the policy store, except the simulated outside_business_hours function returns true.

Additional test cases cover trying to get access to resources with no permissions.



```
----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK
```

2 The Password File

2.1 Hash Function

The Finvest Holdings password system implements the “bcrypt” hashing function.

2.1.1 Hash Length

“bcrpyt” fixes the outputted hash length to 192 characters. This value is not customizable, however, it provides more than enough space to create unique hashes. [1]

$$Hash\ space\ =\ 2^{192}$$

The max input for the hash is also fixed at 72 characters, which is more than enough to create a secure password as well as prevent collisions in the hashing function. [1]

2.1.2 Salt Length

The salt length is fixed at 22 characters and is output in the following format:

$2b$n$abcdefghijklmnopqrstuu.

2b denotes the bcrypt version, n denotes the cost factor of the salt, and the remainder is the 22 character salt used to prepend to the user's password. [1]

As the salt is base-64 encoded [1], the salt/user space is quite a bit larger than the population of earth at:

$$Salt\ space\ =\ 64^{22}$$

2.1.3 Speed

A balance between security and performance was requested by the Finvest Holdings team and bcrypt was selected keeping that in mind. Bcrypt was intentionally designed as a "slow" hash function, for the purpose of deterring potential attackers [1]. The bcrypt salt generation function "gensalt()" takes in a "salt_runs" parameter. This parameter defines the "cost factor" or "work factor" of the hash function. The number ("n") passed into the function means the generation function will run 2^{n} times [1].

By having a customizable parameter for the work factor, the balance between speed and security can be tuned. In the current implementation, the default salt_run is set to 12, but can be adjusted as needed by the system admin. This number was found to be fast enough that users are not affected by runtime, but attackers brute forcing salts would require much more time.

example) Given a compromised password file

$salt\_runs\ =\ 12,\ time\ per\ execute\ =\ 1ms,\ average\ password\ length\ =\ 8$

$character\ space\ =\ 26$

```
iterations = 2 ** 12  # Cost factor
time_per_execution_ms = 1  # Time for one iteration in milliseconds
password_length = 8
possible_characters = 26  # Assuming lowercase letters only


total_time_for_checking_ms = iterations * time_per_execution_ms *
(possible_characters ** password_length)


# Convert to seconds
total_time_for_checking_seconds = total_time_for_checking_ms / 1000

# average time to crack is half the entire time
average_time_to_crack_s = total_time_for_checking_seconds / 2
```

```
#in hours
average_time_to_crack_h = (average_time_to_crack_s / 60) / 60
```

This puts the average time to crack a password in a leaked file at over 118799396 hours.

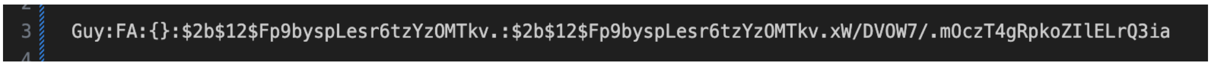
2.2 Password Record Structure

To support RBAC and ABAC with the matrix and policy store, the password file is made up of individual records, each taking up one row. Elements in the record are separated by colons. The structure of each record is as follows:

username:*role*:*attributes*:*salt*:*hash*

Username	The unique identifier of the user
Role	Role user belongs to
Attributes	A key-value representation of attributes belonging to the user and their values
Salt	Generated salt by bcrypt
Hash	The hash of the salted password

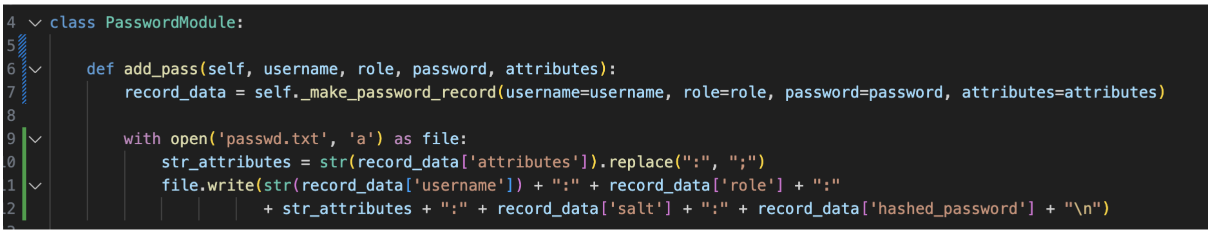
It is important to note that the hash is stored with the salt appended to the front of it, this is just the standard in bcrypt. An example record in this system looks like this:



```
Guy:FA:{}:$2b$12$Fp9byspLesr6tzYzOMTkv.:$2b$12$Fp9byspLesr6tzYzOMTkv.xW/DVOW7/.mOczT4gRpkoZIlELrQ3ia
```

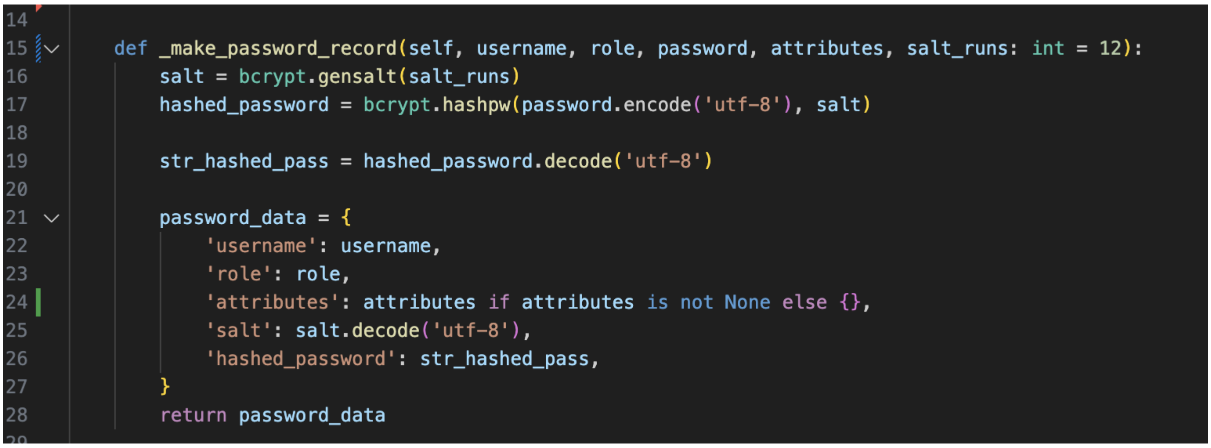
2.3 Implementation

The class PasswordModule is responsible for the addition and checking of password records.



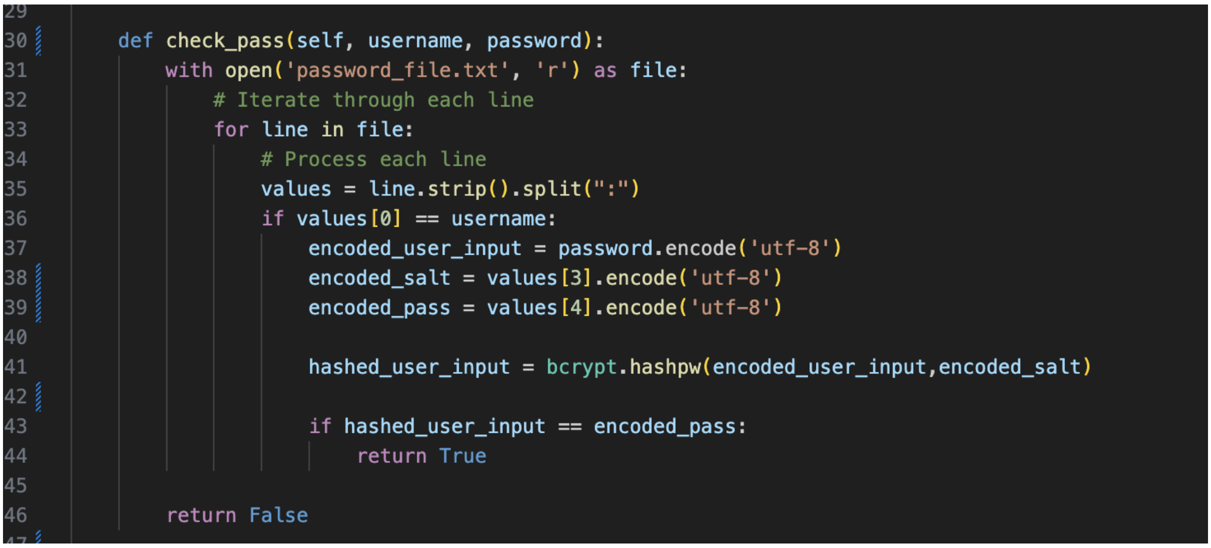
```
class PasswordModule:

    def add_pass(self, username, role, password, attributes):
        record_data = self._make_password_record(username=username, role=role, password=password, attributes=attributes)

        with open('passwd.txt', 'a') as file:
            str_attributes = str(record_data['attributes']).replace(":", ";")
            file.write(str(record_data['username']) + ":" + record_data['role'] + ":"
                       + str_attributes + ":" + record_data['salt'] + ":" + record_data['hashed_password'] + "\n")
```

User information passed to the module is formatted, salts are generated, and passwords are hashed.



```
    def _make_password_record(self, username, role, password, attributes, salt_runs: int = 12):
        salt = bcrypt.gensalt(salt_runs)
        hashed_password = bcrypt.hashpw(password.encode('utf-8'), salt)

        str_hashed_pass = hashed_password.decode('utf-8')

        password_data = {
            'username': username,
            'role': role,
            'attributes': attributes if attributes is not None else {},
            'salt': salt.decode('utf-8'),
            'hashed_password': str_hashed_pass,
        }
        return password_data
```

As this class is responsible for everything to do with the password file, it has methods for determining if a password is correct and if a user exists.

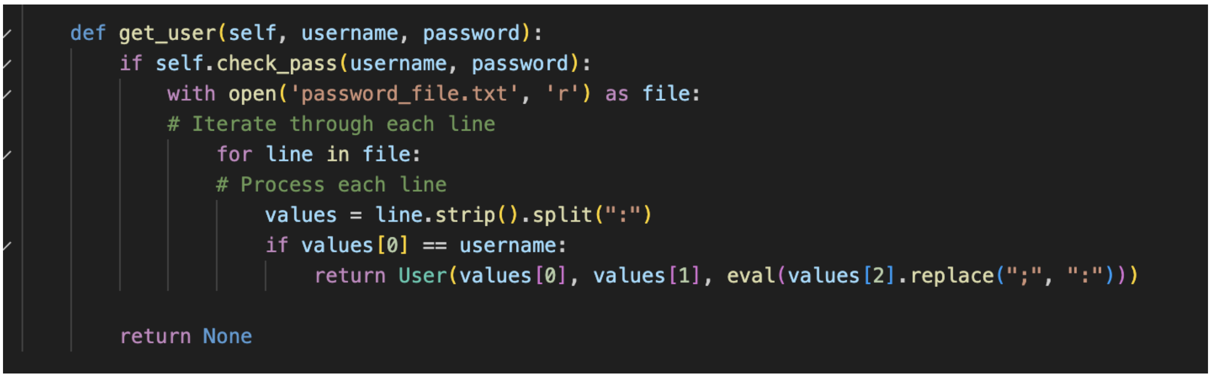


```
    def check_pass(self, username, password):
        with open('password_file.txt', 'r') as file:
            # Iterate through each line
            for line in file:
                # Process each line
                values = line.strip().split(":")
                if values[0] == username:
                    encoded_user_input = password.encode('utf-8')
                    encoded_salt = values[3].encode('utf-8')
                    encoded_pass = values[4].encode('utf-8')

                    hashed_user_input = bcrypt.hashpw(encoded_user_input,encoded_salt)

                    if hashed_user_input == encoded_pass:
                        return True

        return False
```

All the methods that access the password file open it anew rather than reading and storing its information within the program. Although this makes performance marginally worse, it ensures sensitive data is less susceptible to attacks originating in the code (i.e. SQL injections or viruses propagating within system code).

The check_pass() method uses the hashpw() function so that the salt in the password file can be used rather than the stored salt on the hashed password itself. This does not functionally change how the hashing or checking works as the same hash is used either way, it is an implementation detail specific to this assignment.

To retrieve records from the file, a User object is created once determined that it exists, and the password is correct. This User object holds important information about the user: username, role, attributes, but does not include the salt or hashed password. This is again to minimise the surface area of possible attacks.

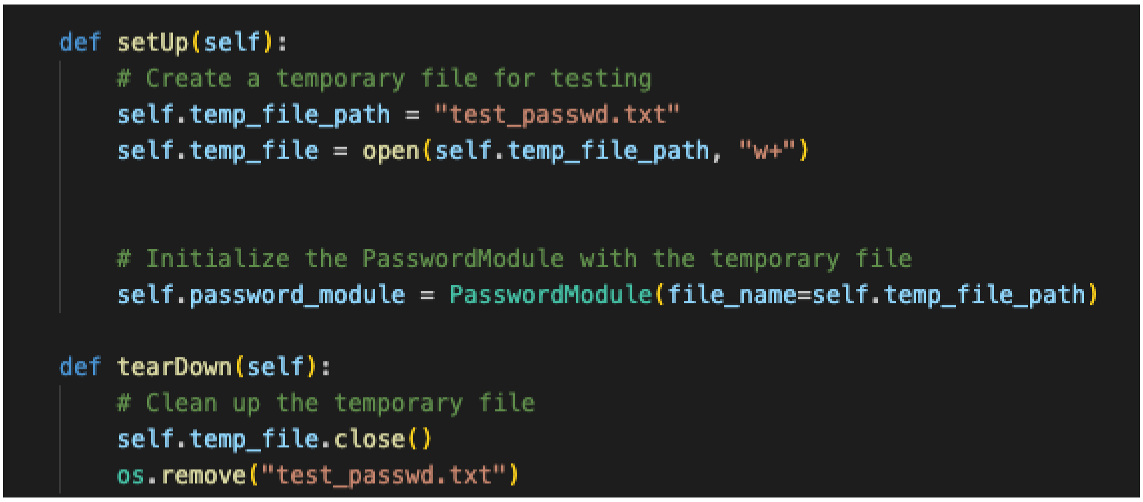


```python
def get_user(self, username, password):
    if self.check_pass(username, password):
        with open('password_file.txt', 'r') as file:
        # Iterate through each line
            for line in file:
            # Process each line
                values = line.strip().split(":")
                if values[0] == username:
                    return User(values[0], values[1], eval(values[2].replace(";", ":")))

    return None
```

2.4 Tests

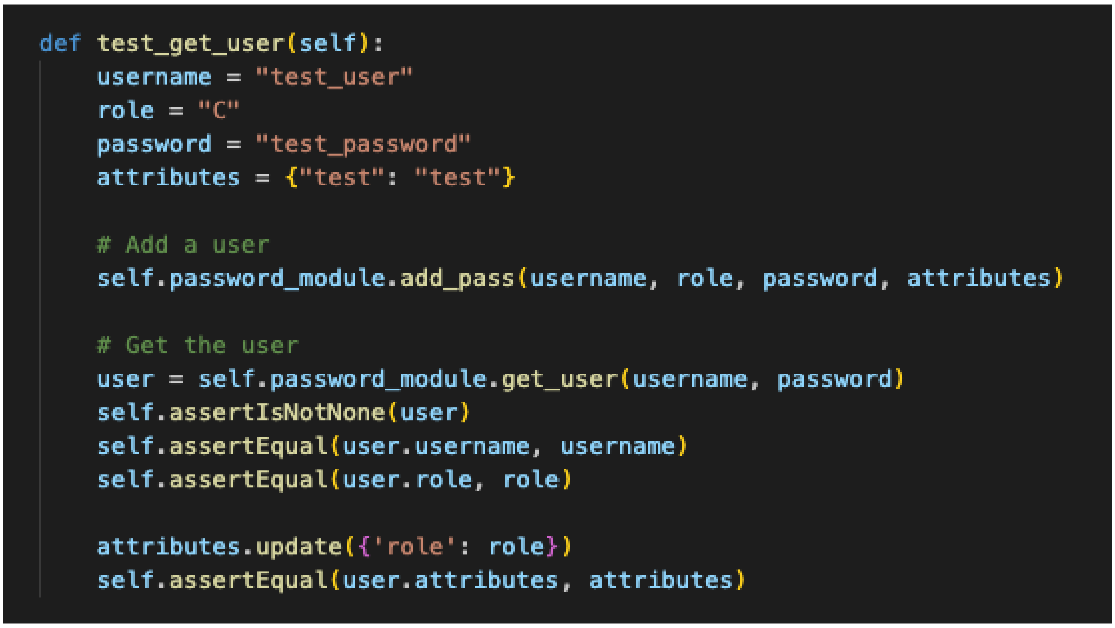
To fully test the password module, each method of the class should be tested. This module doesn’t do much validation on its own, so proactive password checker tests are left for the enrollment module tests.

A temporary file is created at the beginning of the test suite, and deleted at the end.



```python
def setUp(self):
    # Create a temporary file for testing
    self.temp_file_path = "test_passwd.txt"
    self.temp_file = open(self.temp_file_path, "w+")


    # Initialize the PasswordModule with the temporary file
    self.password_module = PasswordModule(file_name=self.temp_file_path)

def tearDown(self):
    # Clean up the temporary file
    self.temp_file.close()
    os.remove("test_passwd.txt")
```

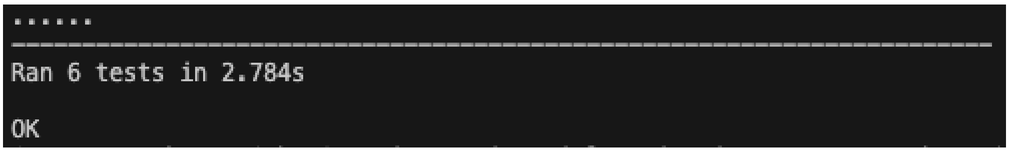
All methods are tested by providing mock data. For example:



```python
def test_get_user(self):
    username = "test_user"
    role = "C"
    password = "test_password"
    attributes = {"test": "test"}

    # Add a user
    self.password_module.add_pass(username, role, password, attributes)

    # Get the user
    user = self.password_module.get_user(username, password)
    self.assertIsNotNone(user)
    self.assertEqual(user.username, username)
    self.assertEqual(user.role, role)

    attributes.update({'role': role})
    self.assertEqual(user.attributes, attributes)
```

Note: Since the User class appends the role to attributes, the dictionary had to be updated to include that change in the tests.

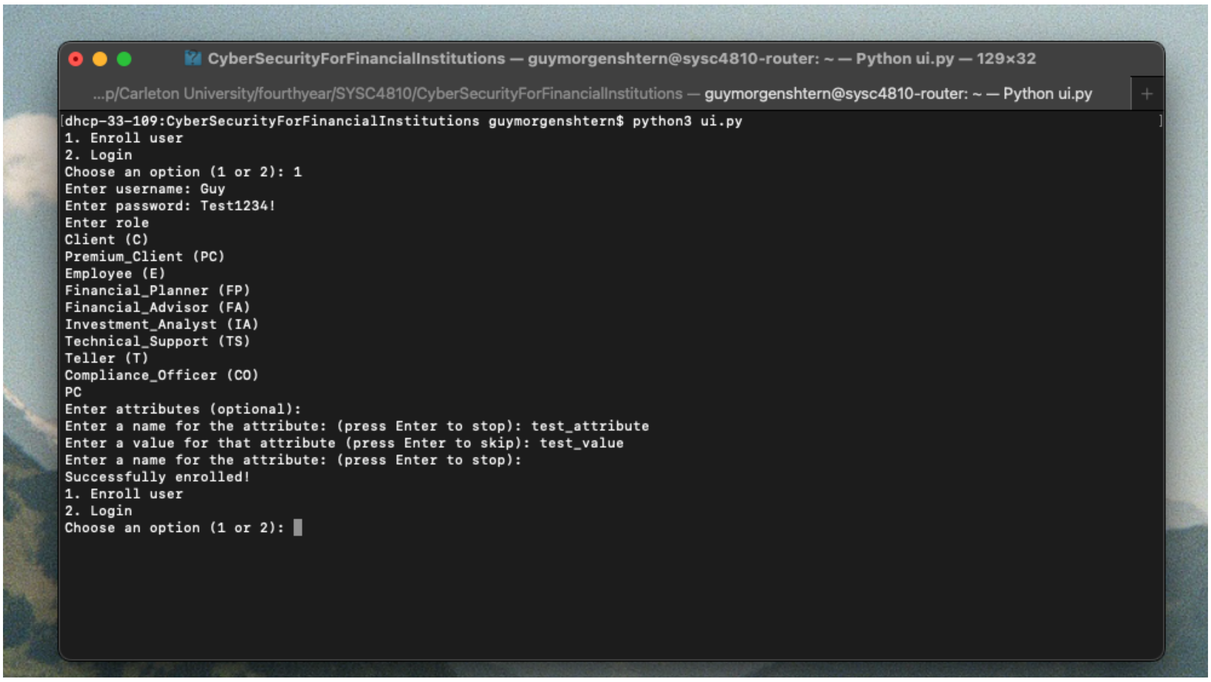
Tests include: get a user, get a user that doesn’t exist, testing is_user when it user exists and doesn’t, adding a password record, checking the added record for accuracy.



```
......
----------------------------------------------------------------------
Ran 6 tests in 2.784s

OK
```

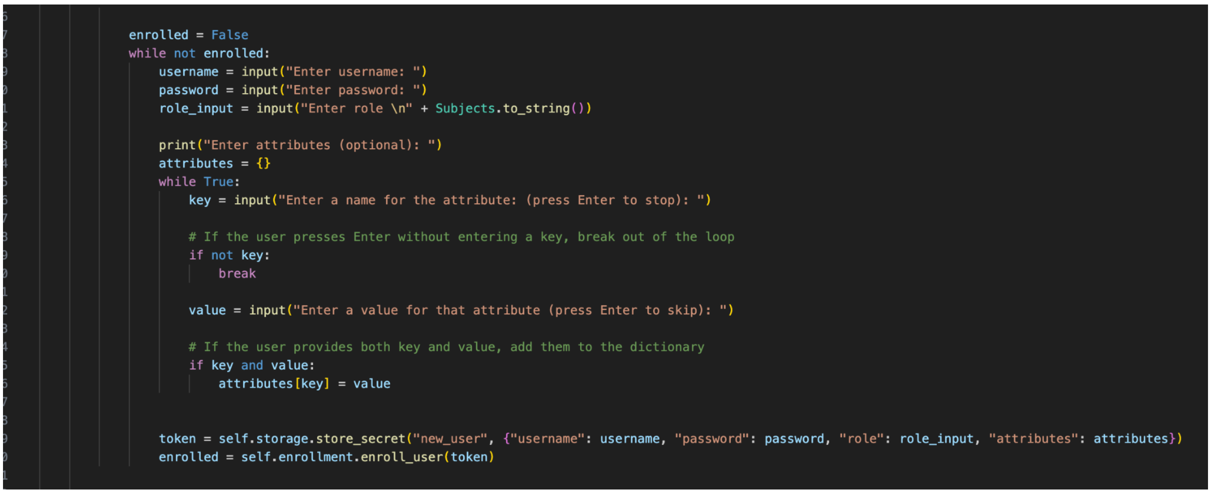
3 User Enrollment

3.1 UI


CyberSecurityForFinancialInstitutions — guymorgenshtern@sysc4810-router: ~ — Python ui.py — 129×32
...p/Carleton University/fourthyear/SYSC4810/CyberSecurityForFinancialInstitutions — guymorgenshtern@sysc4810-router: ~ — Python ui.py

```
dhcp-33-109:CyberSecurityForFinancialInstitutions guymorgenshtern$ python3 ui.py
1. Enroll user
2. Login
Choose an option (1 or 2): 1
Enter username: Guy
Enter password: Test1234!
Enter role
Client (C)
Premium_Client (PC)
Employee (E)
Financial_Planner (FP)
Financial_Advisor (FA)
Investment_Analyst (IA)
Technical_Support (TS)
Teller (T)
Compliance_Officer (CO)
PC
Enter attributes (optional):
Enter a name for the attribute: (press Enter to stop): test_attribute
Enter a value for that attribute (press Enter to skip): test_value
Enter a name for the attribute: (press Enter to stop):
Successfully enrolled!
1. Enroll user
2. Login
Choose an option (1 or 2): 
```

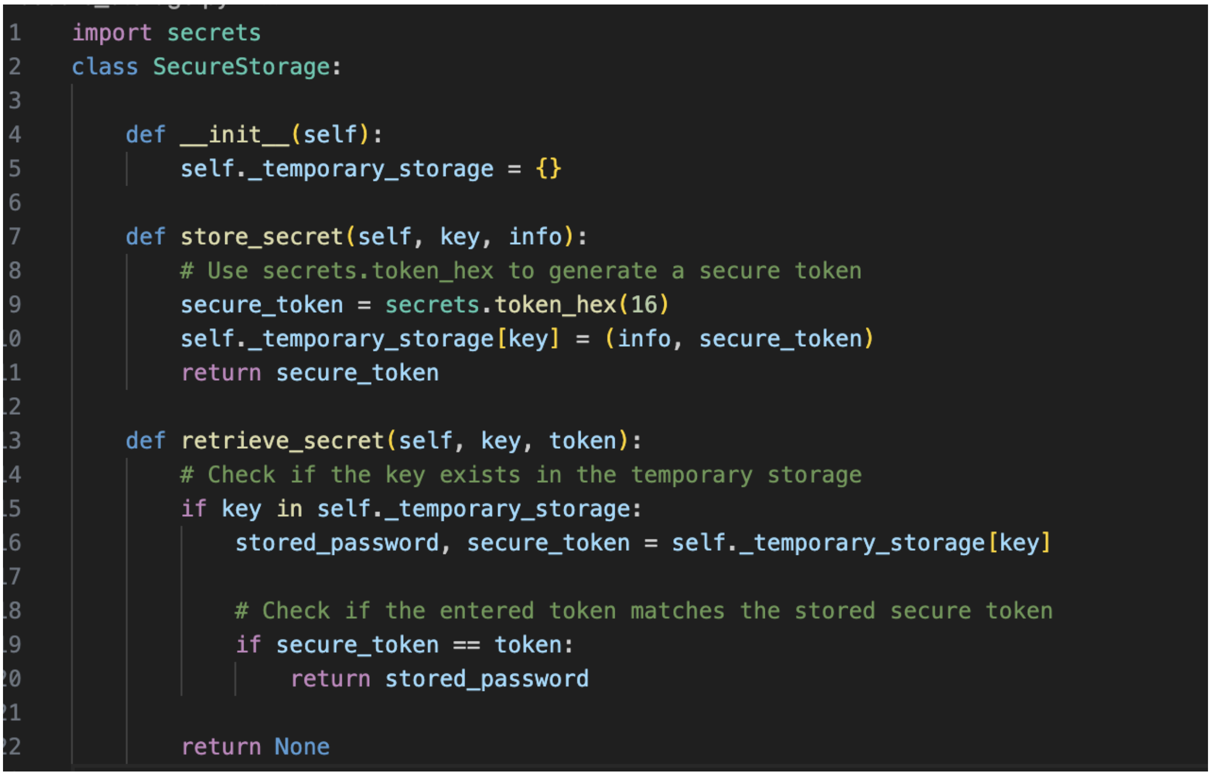
To support RBAC and ABAC, the UI needs to prompt users to provide their role and any additional attributes, along with their username and password.



```
enrolled = False
while not enrolled:
    username = input("Enter username: ")
    password = input("Enter password: ")
    role_input = input("Enter role \n" + Subjects.to_string())

    print("Enter attributes (optional): ")
    attributes = {}
    while True:
        key = input("Enter a name for the attribute: (press Enter to stop): ")

        # If the user presses Enter without entering a key, break out of the loop
        if not key:
            break

        value = input("Enter a value for that attribute (press Enter to skip): ")

        # If the user provides both key and value, add them to the dictionary
        if key and value:
            attributes[key] = value


    token = self.storage.store_secret("new_user", {"username": username, "password": password, "role": role_input, "attributes": attributes})
    enrolled = self.enrollment.enroll_user(token)
```

As attributes are not mandatory, users are given the option to skip that section of enrollment by hitting “Enter”. By clearly walking through the steps of the enrollment process, the user-centric security design principles of “Least Surprise” and “User buy-in” are upheld.

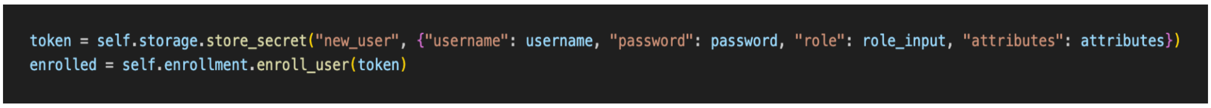
As an additional security measure, whenever users input sensitive data, like their password, data is sent securely using the “secrets” python library.



```python
import secrets
class SecureStorage:

    def __init__(self):
        self._temporary_storage = {}

    def store_secret(self, key, info):
        # Use secrets.token_hex to generate a secure token
        secure_token = secrets.token_hex(16)
        self._temporary_storage[key] = (info, secure_token)
        return secure_token

    def retrieve_secret(self, key, token):
        # Check if the key exists in the temporary storage
        if key in self._temporary_storage:
            stored_password, secure_token = self._temporary_storage[key]

            # Check if the entered token matches the stored secure token
            if secure_token == token:
                return stored_password

        return None
```

This stores information as key-value pairs in a dictionary, but can only be accessed through the “secure_token” which is a cryptographically secure random number.

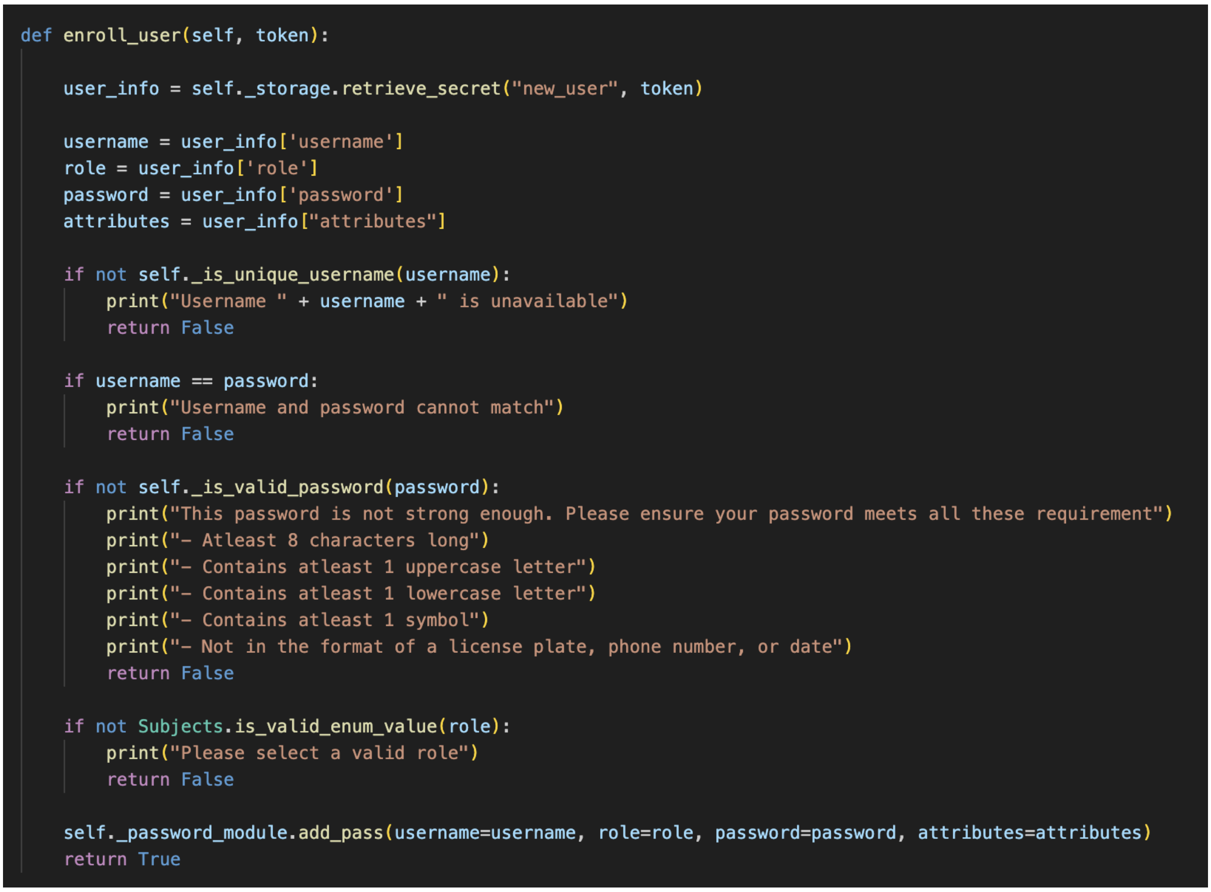
Information is then passed around through the use of the token and a shared SecureStorage object.



```python
token = self.storage.store_secret("new_user", {"username": username, "password": password, "role": role_input, "attributes": attributes})
enrolled = self.enrollment.enroll_user(token)
```

This helps ensure principles like “Security In Depth” and “Isolated Components”.

The enrollment process has additional requirements that must be met: Usernames must be unique, passwords must be deemed valid, usernames cannot include the character “:” as it is the separator used in the password file, and inputted roles must match ones that are in the system. In order to steer users in the right direction, different messages (seen below) are outputted for different errors that help provide more context as to what went wrong with their enrollment process.

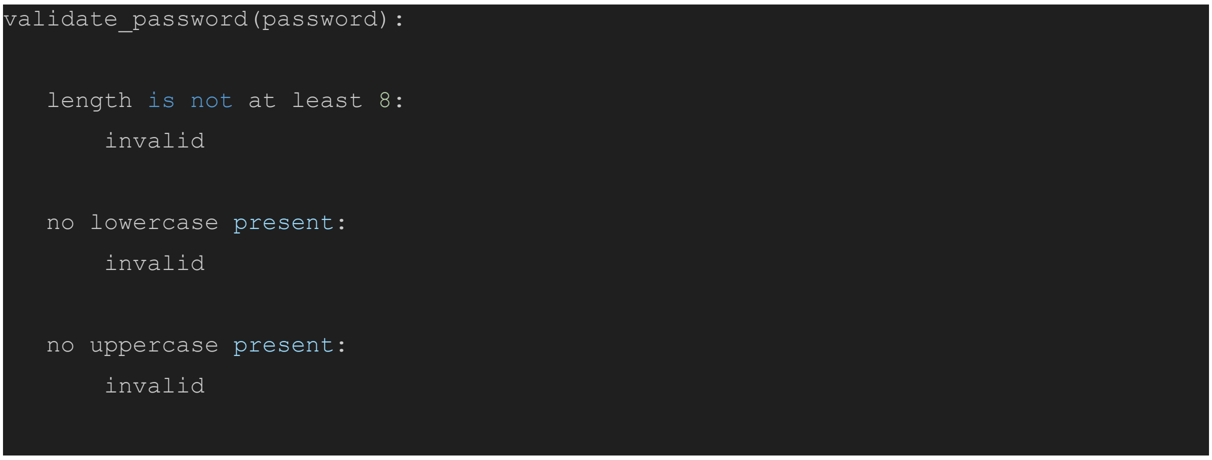


```python
def enroll_user(self, token):

    user_info = self._storage.retrieve_secret("new_user", token)

    username = user_info['username']
    role = user_info['role']
    password = user_info['password']
    attributes = user_info["attributes"]

    if not self._is_unique_username(username):
        print("Username " + username + " is unavailable")
        return False

    if username == password:
        print("Username and password cannot match")
        return False

    if not self._is_valid_password(password):
        print("This password is not strong enough. Please ensure your password meets all these requirement")
        print("- Atleast 8 characters long")
        print("- Contains atleast 1 uppercase letter")
        print("- Contains atleast 1 lowercase letter")
        print("- Contains atleast 1 symbol")
        print("- Not in the format of a license plate, phone number, or date")
        return False

    if not Subjects.is_valid_enum_value(role):
        print("Please select a valid role")
        return False

    self._password_module.add_pass(username=username, role=role, password=password, attributes=attributes)
    return True
```

3.2 Proactive Password Checker

As described by Finvest Holdings, the password must be at least 8 characters, 1 lower, 1 upper, it can’t be in the format of common things like phone numbers, it must include at least 1 digit, and at least 1 special character. Passwords also cannot be the user's username. The system also has a file with 10,000 common passwords. If the inputted password is present in that file, their password is deemed invalid.

Password checking will follow a structure similar to this:

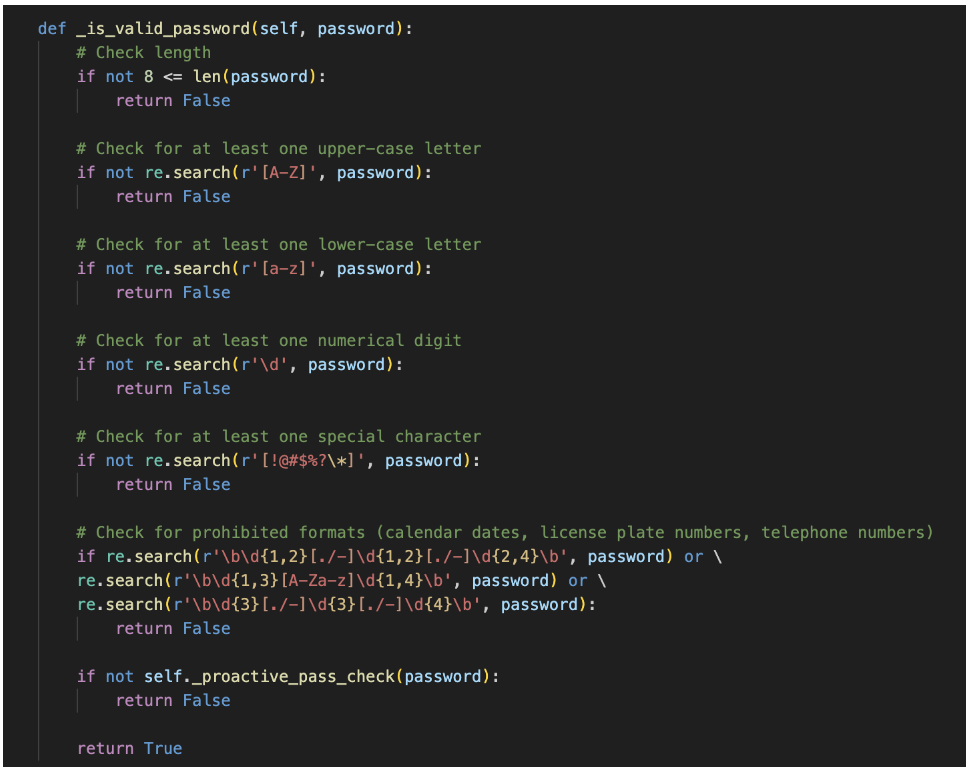


```
validate_password(password):


   length is not at least 8:
       invalid


   no lowercase present:
       invalid


   no uppercase present:
       invalid
```

```
no number present:
    invalid

no special character present:
    invalid

in format of phone number:
    invalid

in format of licence plate:
    invalid

in format of date:
    invalid

is same as username:
    invalid

is password in common_passwords file:
    invalid

otherwise:
    valid
```

3.3 Implementation Proactive Password Checker

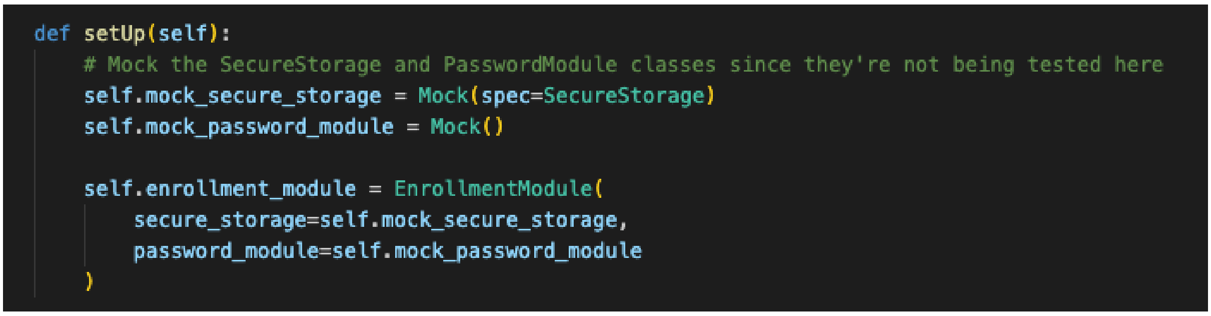
The proactive password checker was designed to encourage strong passwords in users during sign up. It is made up of a few conditionals, all must be met for a password to be deemed valid. The following function determines if the structure of the password is valid:



```
def _is_valid_password(self, password):
    # Check length
    if not 8 <= len(password):
        return False

    # Check for at least one upper-case letter
    if not re.search(r'[A-Z]', password):
        return False

    # Check for at least one lower-case letter
    if not re.search(r'[a-z]', password):
        return False

    # Check for at least one numerical digit
    if not re.search(r'\d', password):
        return False

    # Check for at least one special character
    if not re.search(r'[!@#$%?\*]', password):
        return False

    # Check for prohibited formats (calendar dates, license plate numbers, telephone numbers)
    if re.search(r'\b\d{1,2}[./-]\d{1,2}[./-]\d{2,4}\b', password) or \
    re.search(r'\b\d{1,3}[A-Za-z]\d{1,4}\b', password) or \
    re.search(r'\b\d{3}[./-]\d{3}[./-]\d{4}\b', password):
        return False

    if not self._proactive_pass_check(password):
        return False

    return True
```

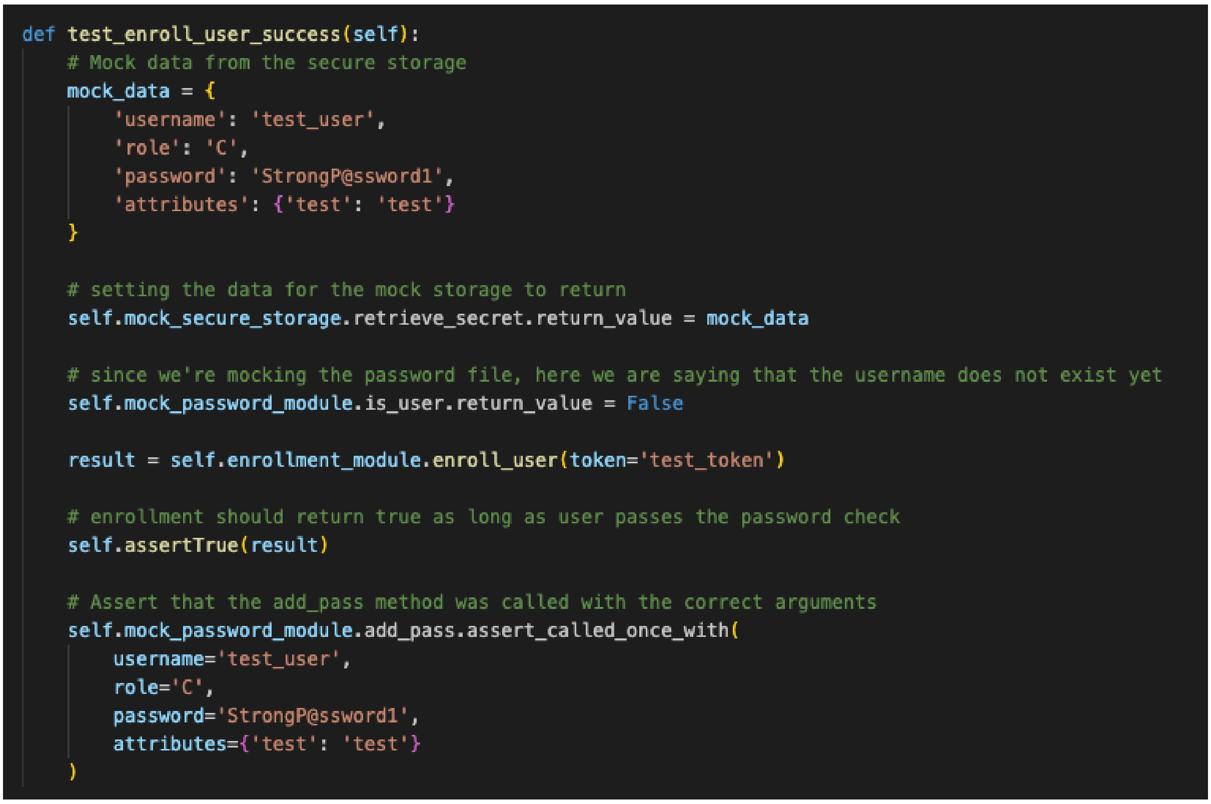
3.4 Tests

The enrollment module heavily relies on the secure storage and password modules. However, it is the enrollment behaviour that needs to be tested specifically. To accomplish that and to limit overhead, Mock objects are used for the storage and password modules.



```python
def setUp(self):
    # Mock the SecureStorage and PasswordModule classes since they're not being tested here
    self.mock_secure_storage = Mock(spec=SecureStorage)
    self.mock_password_module = Mock()

    self.enrollment_module = EnrollmentModule(
        secure_storage=self.mock_secure_storage,
        password_module=self.mock_password_module
    )
```

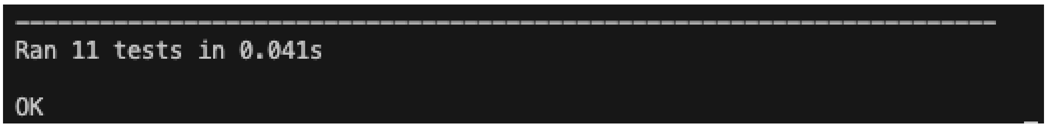
Specific values are attributed to their methods on a test case basis to help simulate behaviour. For example:



```python
def test_enroll_user_success(self):
    # Mock data from the secure storage
    mock_data = {
        'username': 'test_user',
        'role': 'C',
        'password': 'StrongP@ssword1',
        'attributes': {'test': 'test'}
    }

    # setting the data for the mock storage to return
    self.mock_secure_storage.retrieve_secret.return_value = mock_data

    # since we're mocking the password file, here we are saying that the username does not exist yet
    self.mock_password_module.is_user.return_value = False

    result = self.enrollment_module.enroll_user(token='test_token')

    # enrollment should return true as long as user passes the password check
    self.assertTrue(result)

    # Assert that the add_pass method was called with the correct arguments
    self.mock_password_module.add_pass.assert_called_once_with(
        username='test_user',
        role='C',
        password='StrongP@ssword1',
        attributes={'test': 'test'}
    )
```

Here you can see that the test simulates that a user with that username does not exist, then asserts that the correct methods in the password module were called, in this case add_pass().

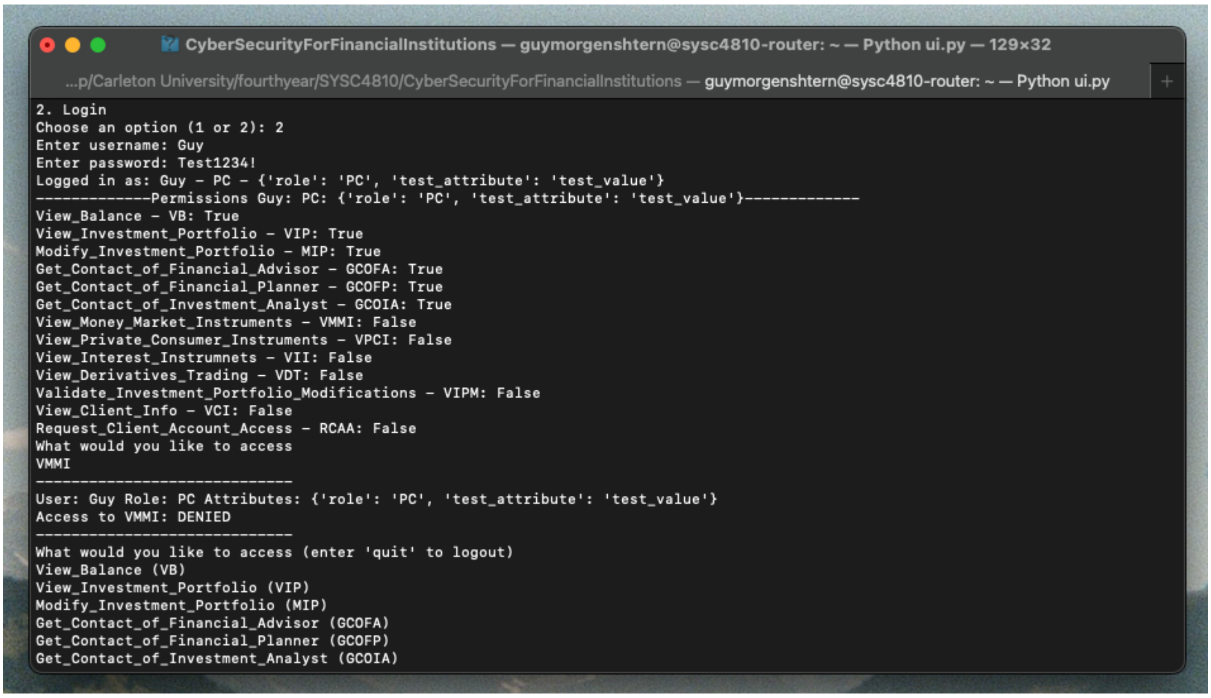
Tests include: Successfully enrolling user, trying to enrol with duplicate username, testing all password checking policies (>=8 characters, >=1 uppercase, >=lowercase, >=1 symbol, >=1 number, no common formats, no passwords present in common password file), and testing a username with invalid character (“:”).



```
----------------------------------------------------------------------
Ran 11 tests in 0.041s

OK
```

4 Login

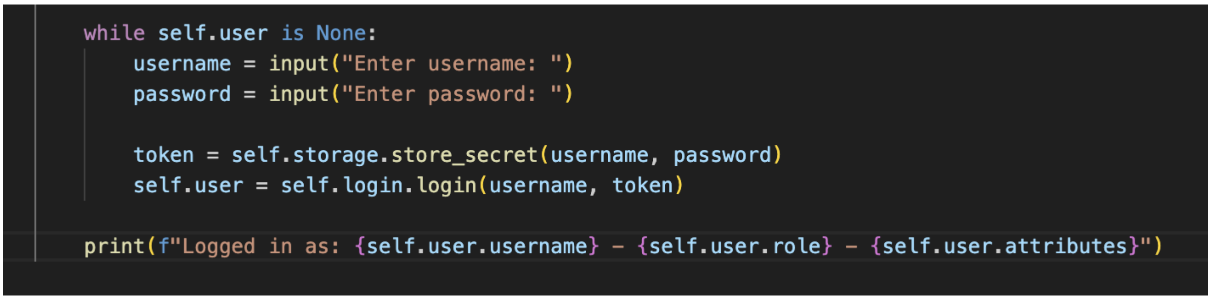
4.1 UI



CyberSecurityForFinancialInstitutions — guymorgenshtern@sysc4810-router: ~ — Python ui.py — 129×32
...p/Carleton University/fourthyear/SYSC4810/CyberSecurityForFinancialInstitutions — guymorgenshtern@sysc4810-router: ~ — Python ui.py

```
2. Login
Choose an option (1 or 2): 2
Enter username: Guy
Enter password: Test1234!
Logged in as: Guy - PC - {'role': 'PC', 'test_attribute': 'test_value'}
-------------Permissions Guy: PC: {'role': 'PC', 'test_attribute': 'test_value'}-------------
View_Balance - VB: True
View_Investment_Portfolio - VIP: True
Modify_Investment_Portfolio - MIP: True
Get_Contact_of_Financial_Advisor - GCOFA: True
Get_Contact_of_Financial_Planner - GCOFP: True
Get_Contact_of_Investment_Analyst - GCOIA: True
View_Money_Market_Instruments - VMMI: False
View_Private_Consumer_Instruments - VPCI: False
View_Interest_Instrumnets - VII: False
View_Derivatives_Trading - VDT: False
Validate_Investment_Portfolio_Modifications - VIPM: False
View_Client_Info - VCI: False
Request_Client_Account_Access - RCAA: False
What would you like to access
VMMI
-----------------------------
User: Guy Role: PC Attributes: {'role': 'PC', 'test_attribute': 'test_value'}
Access to VMMI: DENIED
-----------------------------
What would you like to access (enter 'quit' to logout)
View_Balance (VB)
View_Investment_Portfolio (VIP)
Modify_Investment_Portfolio (MIP)
Get_Contact_of_Financial_Advisor (GCOFA)
Get_Contact_of_Financial_Planner (GCOFP)
Get_Contact_of_Investment_Analyst (GCOIA)
```

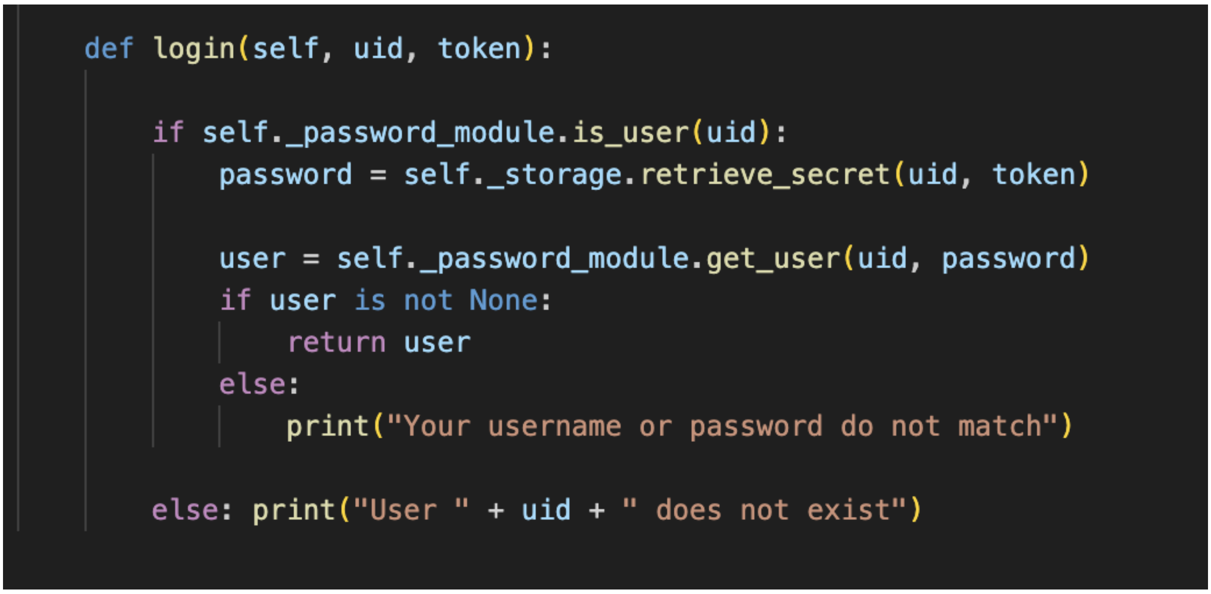
Feedback is given to the user when the login they provide is invalid. “Username does not exist” and “Username and password do not match” provide the user some more information about their credential issues while not compromising security.



```python
while self.user is None:
    username = input("Enter username: ")
    password = input("Enter password: ")

    token = self.storage.store_secret(username, password)
    self.user = self.login.login(username, token)

print(f"Logged in as: {self.user.username} - {self.user.role} - {self.user.attributes}")
```

4.2 Password Verification

Input is taken up by the LoginModule class but actually verification is delegated to the PasswordModule using the methods seen in section 2.



```python
def login(self, uid, token):

    if self._password_module.is_user(uid):
        password = self._storage.retrieve_secret(uid, token)

        user = self._password_module.get_user(uid, password)
        if user is not None:
            return user
        else:
            print("Your username or password do not match")

    else: print("User " + uid + " does not exist")
```

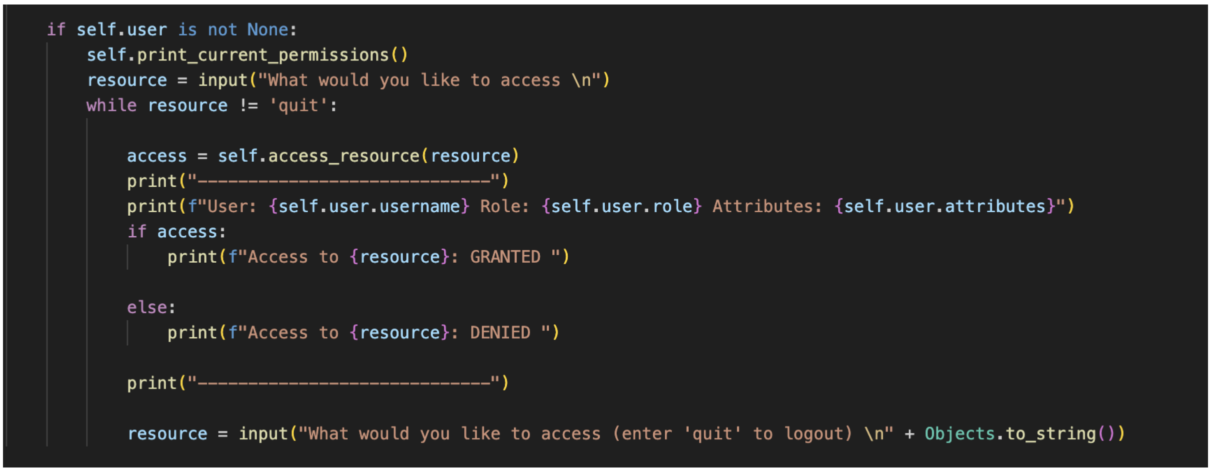
The LoginModule also makes use of SecureStorage since sensitive information is being passed around.

The information is unpacked from storage based on the uid as the key and a token. If the user exists in the password file as defined by the get_user function, then it is returned. Otherwise, the appropriate error message is displayed.

4.3 Resource Access Control

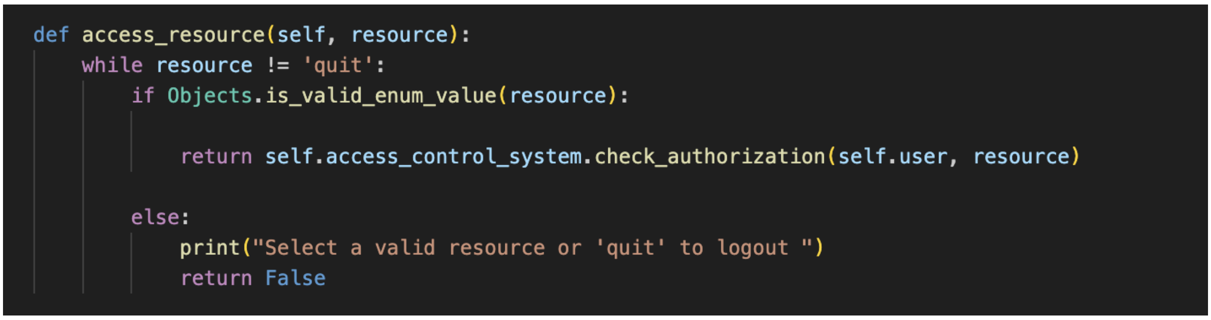
After successfully logging in, users are able to try and access resources in the system.

All the resources are displayed for the users reference. After entering one, users are told if they have access or not, or if the input was invalid, that they need to reenter a resource.



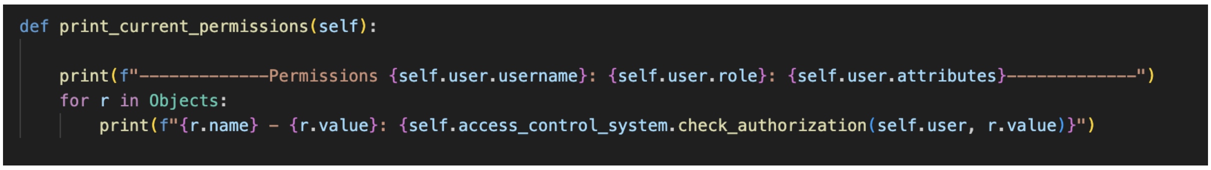
```python
if self.user is not None:
    self.print_current_permissions()
    resource = input("What would you like to access \n")
    while resource != 'quit':

        access = self.access_resource(resource)
        print("-----------------------------")
        print(f"User: {self.user.username} Role: {self.user.role} Attributes: {self.user.attributes}")
        if access:
            print(f"Access to {resource}: GRANTED ")

        else:
            print(f"Access to {resource}: DENIED ")

        print("-----------------------------")

        resource = input("What would you like to access (enter 'quit' to logout) \n" + Objects.to_string())
```

Resources can only begin being accessed when the system detects that a user has successfully been logged in, to ensure safe defaults.



```python
def access_resource(self, resource):
    while resource != 'quit':
        if Objects.is_valid_enum_value(resource):

            return self.access_control_system.check_authorization(self.user, resource)

        else:
            print("Select a valid resource or 'quit' to logout ")
            return False
```

print_current_permissions() outputs the permissions for all resources in the system

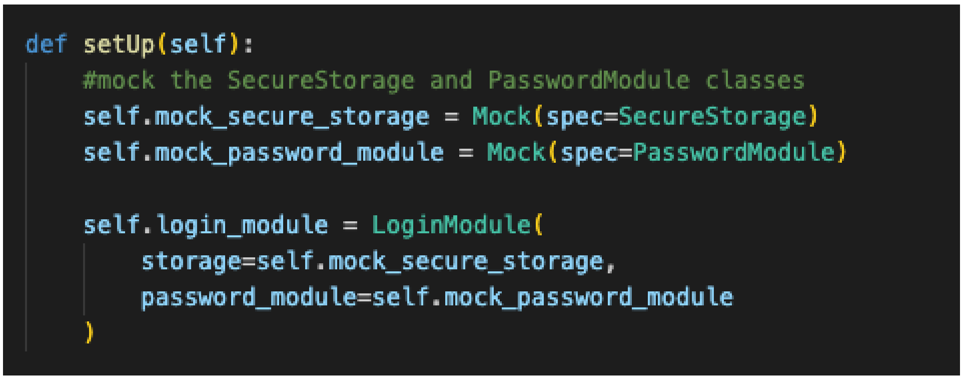


```python
def print_current_permissions(self):

    print(f"------------Permissions {self.user.username}: {self.user.role}: {self.user.attributes}------------")
    for r in Objects:
        print(f"{r.name} - {r.value}: {self.access_control_system.check_authorization(self.user, r.value)}")
```

These methods determine and return the permissions of the currently logged in user. They use the class variable of the user logged in instead of being passed a user to avoid potential spoofing.

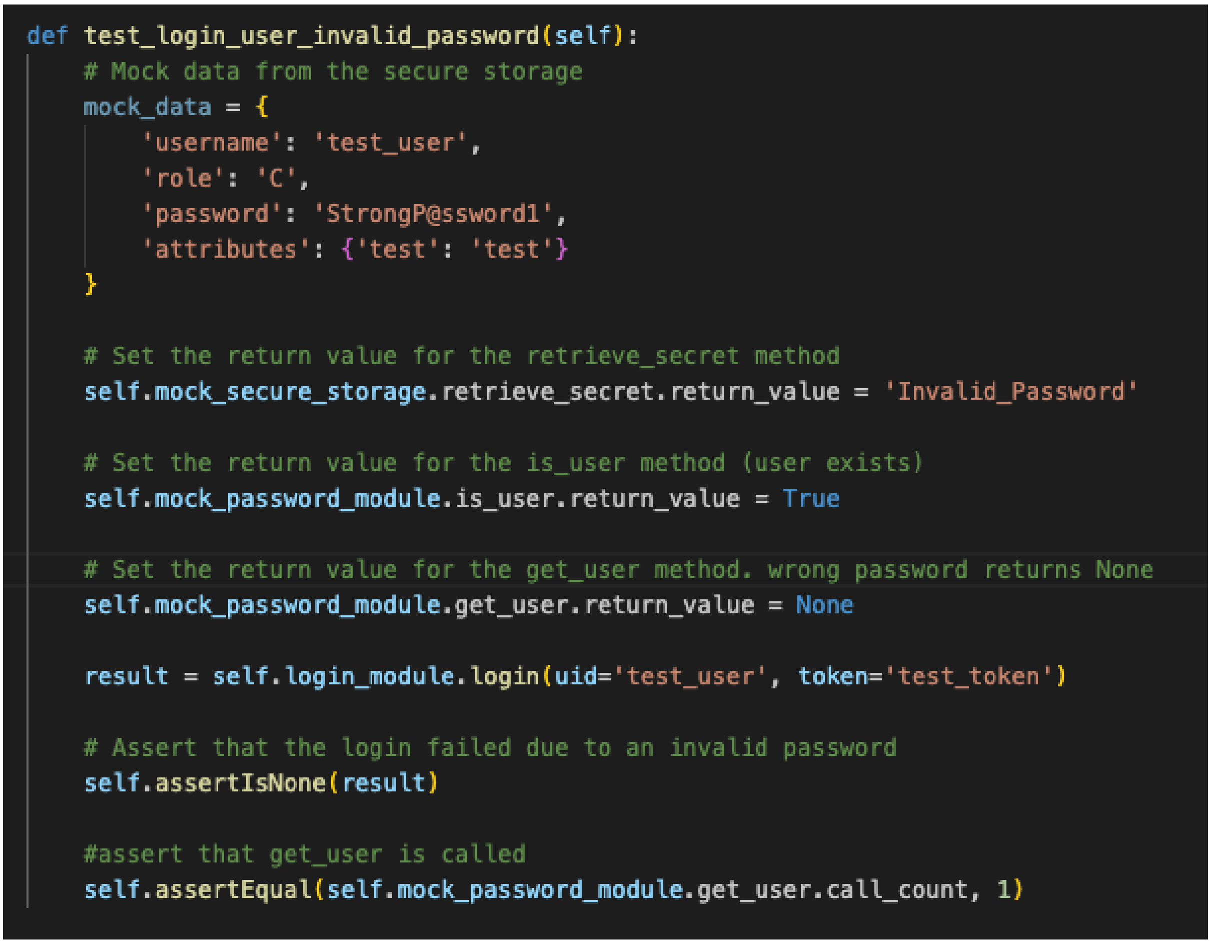
4.4 Tests

Much like the enrollment module, the login module relies on the password and secure storage modules. They are again represented with Mock values.



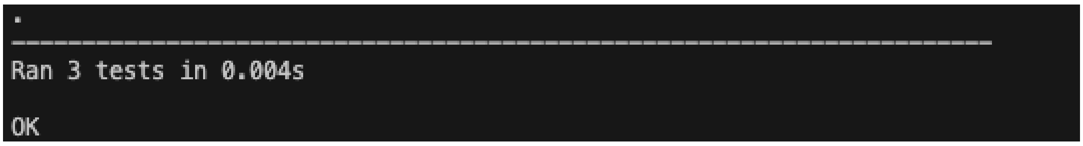
```python
def setUp(self):
    #mock the SecureStorage and PasswordModule classes
    self.mock_secure_storage = Mock(spec=SecureStorage)
    self.mock_password_module = Mock(spec=PasswordModule)

    self.login_module = LoginModule(
        storage=self.mock_secure_storage,
        password_module=self.mock_password_module
    )
```

All values generated from external modules are simulated using the Mock objects. Then, the tests assert that the correct values are returned, and if applicable, the correct internal method calls were made.



```
def test_login_user_invalid_password(self):
    # Mock data from the secure storage
    mock_data = {
        'username': 'test_user',
        'role': 'C',
        'password': 'StrongP@ssword1',
        'attributes': {'test': 'test'}
    }

    # Set the return value for the retrieve_secret method
    self.mock_secure_storage.retrieve_secret.return_value = 'Invalid_Password'

    # Set the return value for the is_user method (user exists)
    self.mock_password_module.is_user.return_value = True

    # Set the return value for the get_user method. wrong password returns None
    self.mock_password_module.get_user.return_value = None

    result = self.login_module.login(uid='test_user', token='test_token')

    # Assert that the login failed due to an invalid password
    self.assertIsNone(result)

    #assert that get_user is called
    self.assertEqual(self.mock_password_module.get_user.call_count, 1)
```

Tests include: Successful login, unsuccessful login due to wrong password, and unsuccessful login due to username not existing.



```
.
----------------------------------------------------------------------
Ran 3 tests in 0.004s

OK
```

5 Summary

The Finvest Holdings User Authorization and Authentication System uses a combination RBAC and ABAC authorization policy implemented with an Access Control Matrix and Policy Store respectively. The combination provides the first steps to ensuring proper balance between security and performance. With the matrix, broad roles can be defined with their permissions, new ones can be added simply by appending to the "matrix.txt" file. The policy store allows for fine grained control over permission restriction. Custom conditions can be written and added to the store easily through the use of the add_policy() function. These policies can include custom attributes, which do not have to be present on each user in order to function and environment variables which can be optionally specified.

To further provide balance between performance and security, "bcrypt" was selected as the hashing function for this system. As a "slow" hashing function, it naturally deters potential attackers. However, the additional tuning of the work function allows for the system's security to keep up with improving computation speeds. The default work function is 12, which is fast enough that users do not notice a delay, but greatly increases resource and time requirements for brute force attackers. As a note, this system provides support for passwords up to 72 characters in length [1].

For added security and a better user experience, this system is equipped to detect and deny weak passwords from being used in enrollment and login. Users are walked through the various requirements for a password and specific messages guide them through mistakes in their password creation process.

Program Flow/Usecase

1. **In terminal, navigate into parent folder**

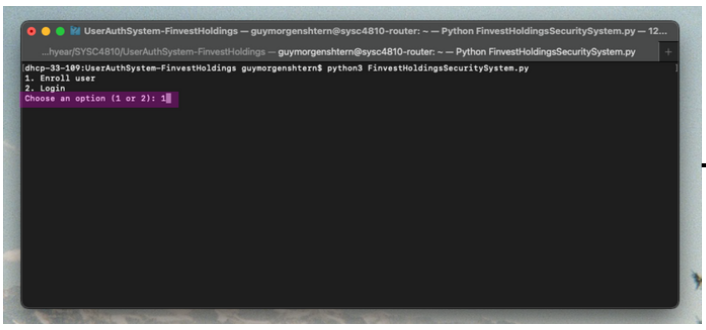
```
cd path/to/assignment/UserAuthSystem-FinvestHoldings
```

2. **In terminal, run**

```
python3 FinvestHoldingsSecuritySystem.py
```

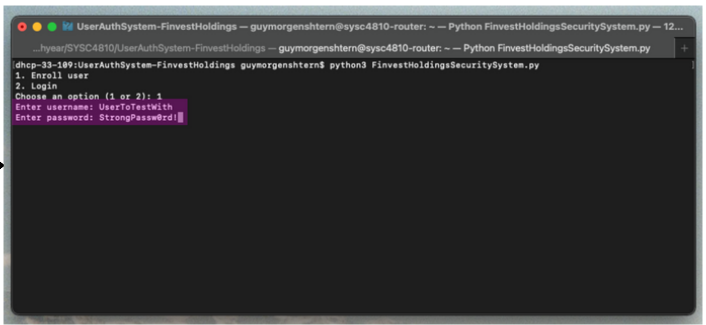
3. **Follow the 'enrollment' workflow**

Input '1'


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

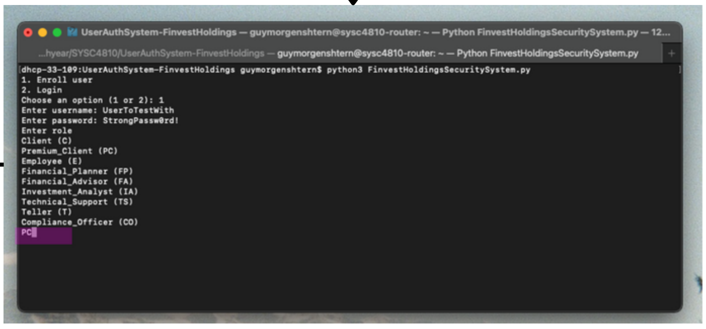
```
dhcp-33-109:UserAuthSystem-FinvestHoldings guymorgenshtern$ python3 FinvestHoldingsSecuritySystem.py
1. Enroll user
2. Login
Choose an option (1 or 2): 1
```

Set a username and password


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

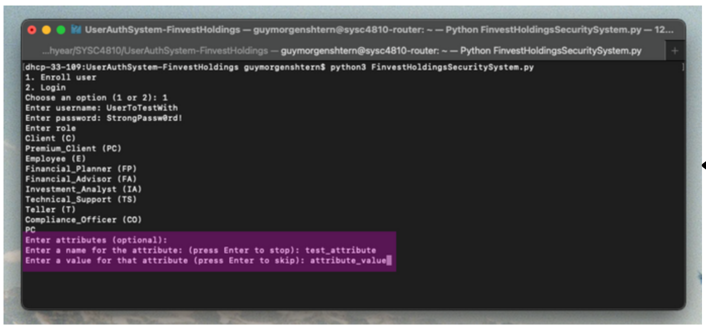
```
dhcp-33-109:UserAuthSystem-FinvestHoldings guymorgenshtern$ python3 FinvestHoldingsSecuritySystem.py
1. Enroll user
2. Login
Choose an option (1 or 2): 1
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
```

Input a role


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

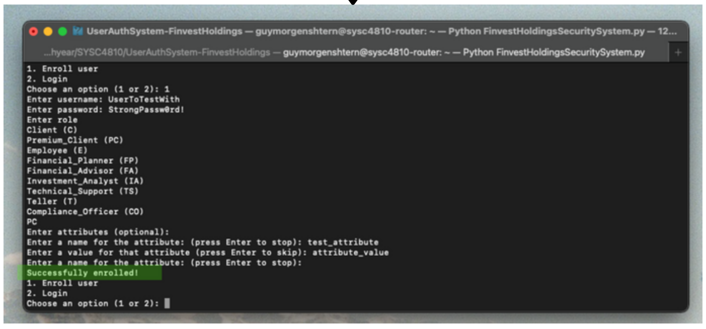
```
dhcp-33-109:UserAuthSystem-FinvestHoldings guymorgenshtern$ python3 FinvestHoldingsSecuritySystem.py
1. Enroll user
2. Login
Choose an option (1 or 2): 1
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
Enter role
Client (C)
Premium_Client (PC)
Employee (E)
Financial_Planner (FP)
Financial_Advisor (FA)
Investment_Analyst (IA)
Technical_Support (TS)
Teller (T)
Compliance_Officer (CO)
PC
```

Add attributes (optional, 'enter' to skip)


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

```
dhcp-33-109:UserAuthSystem-FinvestHoldings guymorgenshtern$ python3 FinvestHoldingsSecuritySystem.py
1. Enroll user
2. Login
Choose an option (1 or 2): 1
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
Enter role
Client (C)
Premium_Client (PC)
Employee (E)
Financial_Planner (FP)
Financial_Advisor (FA)
Investment_Analyst (IA)
Technical_Support (TS)
Teller (T)
Compliance_Officer (CO)
PC
Enter attributes (optional):
Enter a name for the attribute: (press Enter to stop): test_attribute
Enter a value for that attribute (press Enter to skip): attribute_value
```

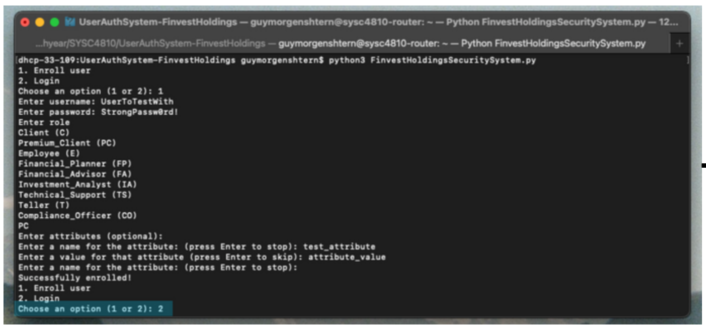
Success!


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

```
1. Enroll user
2. Login
Choose an option (1 or 2): 1
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
Enter role
Client (C)
Premium_Client (PC)
Employee (E)
Financial_Planner (FP)
Financial_Advisor (FA)
Investment_Analyst (IA)
Technical_Support (TS)
Teller (T)
Compliance_Officer (CO)
PC
Enter attributes (optional):
Enter a name for the attribute: (press Enter to stop): test_attribute
Enter a value for that attribute (press Enter to skip): attribute_value
Enter a name for the attribute: (press Enter to stop):
Successfully enrolled!
1. Enroll user
2. Login
Choose an option (1 or 2):
```

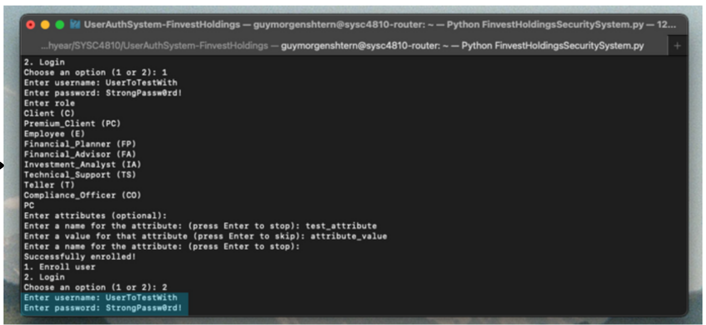
4. **Follow the 'login' workflow**

Input '2'


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

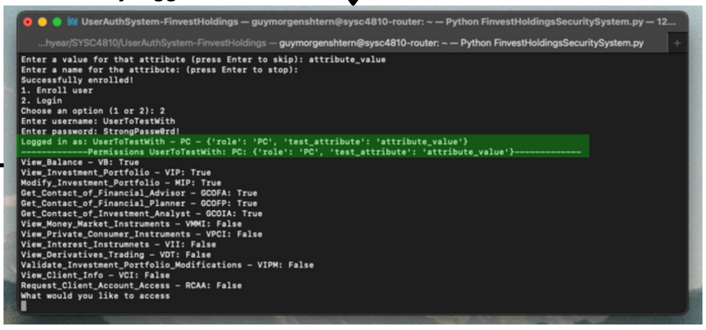
```
dhcp-33-109:UserAuthSystem-FinvestHoldings guymorgenshtern$ python3 FinvestHoldingsSecuritySystem.py
1. Enroll user
2. Login
Choose an option (1 or 2): 1
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
Enter role
Client (C)
Premium_Client (PC)
Employee (E)
Financial_Planner (FP)
Financial_Advisor (FA)
Investment_Analyst (IA)
Technical_Support (TS)
Teller (T)
Compliance_Officer (CO)
PC
Enter attributes (optional):
Enter a name for the attribute: (press Enter to stop): test_attribute
Enter a value for that attribute (press Enter to skip): attribute_value
Enter a name for the attribute: (press Enter to stop):
Successfully enrolled!
1. Enroll user
2. Login
Choose an option (1 or 2): 2
```

Input username and password


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

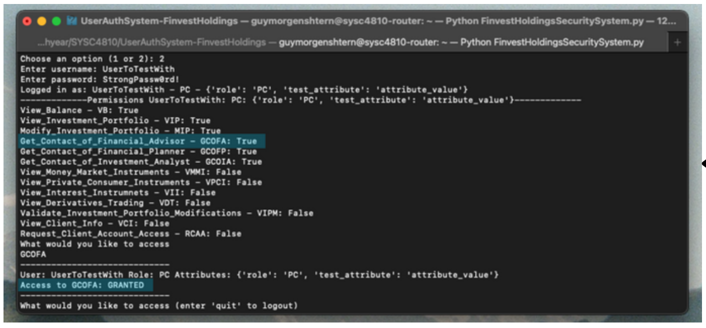
```
2. Login
Choose an option (1 or 2): 1
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
Enter role
Client (C)
Premium_Client (PC)
Employee (E)
Financial_Planner (FP)
Financial_Advisor (FA)
Investment_Analyst (IA)
Technical_Support (TS)
Teller (T)
Compliance_Officer (CO)
PC
Enter attributes (optional):
Enter a name for the attribute: (press Enter to stop): test_attribute
Enter a value for that attribute (press Enter to skip): attribute_value
Enter a name for the attribute: (press Enter to stop):
Successfully enrolled!
1. Enroll user
2. Login
Choose an option (1 or 2): 2
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
```

Successfully logged in!


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

```
Enter a value for that attribute (press Enter to skip): attribute_value
Enter a name for the attribute: (press Enter to stop):
Successfully enrolled!
1. Enroll user
2. Login
Choose an option (1 or 2): 2
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
Logged in as: UserToTestWith - PC - {'role': 'PC', 'test_attribute': 'attribute_value'}
-------------Permissions UserToTestWith: PC: {'role': 'PC', 'test_attribute': 'attribute_value'}-------------
View_Balance - VB: True
View_Investment_Portfolio - VIP: True
Modify_Investment_Portfolio - MIP: True
Get_Contact_of_Financial_Advisor - GCOFA: True
Get_Contact_of_Financial_Planner - GCOFP: True
Get_Contact_of_Investment_Analyst - GCOIA: True
View_Money_Market_Instruments - VMMI: False
View_Private_Consumer_Instruments - VPCI: False
View_Interest_Instrumnets - VII: False
View_Derivatives_Trading - VDT: False
Validate_Investment_Portfolio_Modifications - VIPM: False
View_Client_Info - VCI: False
Request_Client_Account_Access - RCAA: False
What would you like to access
```

View permissions for resources by inputting name


UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py — 12...
...hyear/SYSC4810/UserAuthSystem-FinvestHoldings — guymorgenshtern@sysc4810-router: ~ — Python FinvestHoldingsSecuritySystem.py

```
Choose an option (1 or 2): 2
Enter username: UserToTestWith
Enter password: StrongPassw0rd!
Logged in as: UserToTestWith - PC - {'role': 'PC', 'test_attribute': 'attribute_value'}
-------------Permissions UserToTestWith: PC: {'role': 'PC', 'test_attribute': 'attribute_value'}-------------
View_Balance - VB: True
View_Investment_Portfolio - VIP: True
Modify_Investment_Portfolio - MIP: True
Get_Contact_of_Financial_Advisor - GCOFA: True
Get_Contact_of_Financial_Planner - GCOFP: True
Get_Contact_of_Investment_Analyst - GCOIA: True
View_Money_Market_Instruments - VMMI: False
View_Private_Consumer_Instruments - VPCI: False
View_Interest_Instrumnets - VII: False
View_Derivatives_Trading - VDT: False
Validate_Investment_Portfolio_Modifications - VIPM: False
View_Client_Info - VCI: False
Request_Client_Account_Access - RCAA: False
What would you like to access
GCOFA
-----------------------------
User: UserToTestWith Role: PC Attributes: {'role': 'PC', 'test_attribute': 'attribute_value'}
Access to GCOFA: GRANTED
-----------------------------
What would you like to access (enter 'quit' to logout)
```

5. **Enter 'quit' to logout**

References

[1] “Bcrypt,” PyPI, https://pypi.org/project/bcrypt/ (accessed Dec. 3, 2023).