[H-1] Secret `s_password` is stored on chain and visible to anyone

Description The private variable `s_password` is intended to only be accessible to the owner of the contract through the `getPassword()` method, however all data submitted to the blockchain can be accessed by anyone.

Impact This allows anyone to read the password, breaking the functionality of the protocol and leaking sensitive information.

Proof of Concept The following proof of concept can be followed to confirm this issue, note you will need foundry.

1. Create a local blockchain using anvil

```
make anvil
```

2. Deploy the contract to the chain.

```
make deploy
```

3. Use cast to find the storage item at storage slot 1.

```
cast storage <CONTRACT_ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

This will return a hex representation of the `s_password` secret.

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Finally we can get the string representation of `s_password` using the following.

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

This will return the string representation of `s_password` which is:

```
myPassword
```

Recommended Mitigation Due to this finding, the overall architecture of the contract should be rethought. You could encrypt the password offchain and then store the encrypted password onchain. However, the user would then need to remember a second password to decrypt the encrypted password.

[H-2] `setPassword()` has no access control, allowing non-owner to set the password

Description The `setPassword()` lacks access control, however, according to the natspec is should only be callable by the owner: `This function allows only the owner to set a new password`.

```
@>     function setPassword(string memory newPassword) external { // @audit - There is no access control
           s_password = newPassword;
           emit SetNetPassword();
       }
```

Impact This allows anyone to set the password, breaking the invariant that only the owner should be able to set the password.

Proof of Concept The following can be added to your test suite in `tests/PasswordStore.t.sol` to verify this issue:

```
    function test_non_owner_setting_password_reverts() public {
        vm.startPrank(vm.addr(1));

        vm.expectRevert(PasswordStore.PasswordStore__NotOwner.selector)
            ;
        passwordStore.setPassword("nonOwnerPassword");
    }
```

Now when running the test suite with `forge test` we get one failing test with the following output:

```
Failing tests:
Encountered 1 failing test in test/PasswordStore.t.sol:
    PasswordStoreTest
[FAIL. Reason: call did not revert as expected]
    test_non_owner_setting_password_reverts() (
gas: 15587)
```

Recommended Mitigation Copy the access control check from `getPassword()` to `setPassword()` as follows:

```
    function setPassword(string memory newPassword) external {
+       if (msg.sender != s_owner) {
+           revert PasswordStore__NotOwner();
+       }

        s_password = newPassword;
        emit SetNetPassword();
    }
```

Once this is added, all tests, including the one added in the proof of concept, pass.

[I-1] The `getPassword()` natspec includes a incorrect parameter

Description The natspec for `getPassword()` is incorrect as it states that the function takes `newPassword` as a parameter, which it does not, it takes no parameters.

```
    /*
     * @notice This allows only the owner to retrieve the password.
@>   * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

Impact The natspec is incorrect.

Recommended Mitigation Remove the incorrect line from the natspec.

```
-     * @param newPassword The new password to set.
```