

[S-1] Storing password on-chain makes it visible to anyone

Description: All data stored on chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

Impact: Anyone can read the private password, breaking functionality of the protocol.

Proof of Concept:

The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

This will default to your local node. You need to have it running in another terminal in order for it to deploy.

```
make deploy
```

3. Run the storage tool using this local chain and rpc-url

This will default to your local node. You need to have it running in another terminal in order for it to deploy.

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http://127.0.0.1:8545
```

4. Read password

```
cast parse-bytes32-string 0x6d7950617373776f72640000000000000000000000000000000000000000000014
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off chain and the store the encrypted password on-chain. But this would require the user to know 2 passwords instead of just 1.

[S-2] The `PasswordStore::setPassword` has no access control, meaning a non-owner could change the password.

Description: The `PasswordStore::setPassword` function is set to be an external function, however, the natspec of the function and overall purpose is This allows only the owner to retrieve the password.

Impact: Anyone can set/change the password of the contract, breaking contract functionality.

Proof of Concept: Add the following to the PasswordStore.t.sol test file.

```
function test_anyone_can_set_password(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.prank(randomAddress);
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();
    assertEq(actualPassword, expectedPassword);
}
```

Recommended Mitigation: Add an access control conditional to the setPassword function.

```
if(msg.sender != s_owner) {
    revert PasswordStore__NotOwner();
}
```