

[H-1] Storing the password on-chain makes it visible to anyone, and no longer private

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 called only by the owner of the contract.

We show one such method of reading any data off-chain below.

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

Proof of Concept: (Proof of Code)

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

- ## 1. Create a locally running chain

```
1 make anvil
```

- ## 2. Deploy the contract to the chain

```
1 make deploy
```

- ### 3. Run the storage tool

```
1 cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

You get a similar output: 0x6d7950617373776f726400

You can now parse that hex to a string with:

[illegible]

And get an output of:

```
1 myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

Likelihood & Impact:

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

[H-2] PasswordStore::setPassword has no access controls, 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 of the smart contract is that This function allows only the owner to set a `new` password.

```
1 function setPassword(string memory newPassword) external {
2     // @audit There are no access controls
3     s_password = newPassword;
4     emit SetNetPassword();
5 }
```

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

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

Code

```
1 function test_anyone_can_set_password(address randomAddress) public {
2     vm.assume(randomAddress != owner);
3     vm.prank(randomAddress);
4     string memory expectedPassword = "myNewPassword";
5     passwordStore.setPassword(expectedPassword);
6
7     vm.prank(owner);
8     string memory actualPassword = passwordStore.getPassword();
9     assertEq(actualPassword, expectedPassword);
10 }
```

Recommended Mitigation: Add a conditional to the `setPassword` function.

```
1 if(msg.sender != s_owner){
2     revert PasswordStorage_NotOwner();
3 }
```

Likelihood & Impact:

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

[I-1] The PasswordStore::getPassword natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
1      /*
2      * @notice This allows only the owner to retrieve the password.
3      * @param newPassword The new password to set.
4      */
5
6      // @audit there is no param to be passed (documentation fix)
7      function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()` which the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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

Likelihood & Impact:

- Impact: NONE
- Likelihood: HIGH
- Severity: Informational/Gas/Non-crits