Audit Report - PasswordStore

`Title` : `PasswordStore Audit Report`

`Author` : `DHANANJAY BHAVAR`

`Date` : `December 29, 2023`

Protocol Summary

A smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Risk Classification

`High`: Direct impact on the funds or the main functionality of the protocol.

`Medium`: Indirect impact on the funds or the protocol's functionality.

`Low`: Minimal to no impact on the funds or the protocol's main functionality.

`Informational`: Suggestions related to good coding practices and gas-efficient code.

Audit Scope

`commit hash`: 7d55682ddc4301a7b13ae9413095feffd9924566

`contracts`: ./src/PasswordStore.sol

`Solc Version`: 0.8.18

`Chain(s)`: Ethereum

[H-1] Password stored on-chain is visible to everyone regardless of the solidity visibility keyword is used.

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

Impact: Anyone can read the 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 from the blockchain.
Let's go over how we programmatically prove the claim we're making - that anyone can read the protocol's stored password.
First we need a local chain running.
forge anvil

Foundry allows us to check the storage of a deployed contract with a very simple cast command. For this we'll need to recall to which storage slot the s_password variable is assigned.

With this consideration we can run the command cast storage <address> <storageSlot> like this (your address may be different).
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1

We should receive an output similar to this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

This is the bytes form of the data at storage slot 1. By using another convenient Foundry command we can now decode this data.
cast parse-bytes32-string
0x6d7950617373776f726400000000000000000000000000000000000000000014

Our output then becomes:
myPassword

Recommended Mitigation: Overall architecture needs to be modified to mitigate the above vulnerability. We can encrypt the password off chain and then store the encrypted password on the blockchain. This will lead to the original password to be safe. User should remember another key off chain to decrypt the password. However you will also like to remove the view function as it would lead to issues in case the user accidently exposes the decryption key.

Likelihood & Impact :

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

[H-1] `PasswordStore::set_Password` has no access controls, meaning a non-owner could change or set the password as well.

Description: There is no check in the `PasswordStore::set_Password` function to actually verify that its actually the owner who's calling the function. Therefore due to this anyone can change or set the password and the main purpose of the contract is compromised.

Impact: This will lead to anyone changing/setting the users password breaking the contracts intended functionality.

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

```
function test_non_owner_can_set_password(address randomAddress)
        vm.assume(owner != randomAddress);
        vm.prank(randomAddress);
        string memory randomUserPass = "randomPassword";
        passwordStore.setPassword(randomUserPass);
```

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

The above test runs successfully for various values of random addresses proving the mentioned bug.

Recommended Mitigation: Add an access control condition at the beginning in the `setPassword` function.

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

Likelihood & Impact :

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

[I-1] The comment before `PasswordStore.sol::getPassword` function mentions a parameter named `newPassword` that doesn't exist in the function, making the comment to be incorrect.

Description: There is no such parameter named `newPassword` used in the `getPassword` function which is stated in comments specifying the working of the function.

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

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

Impact: Reduce the readability of the contract.

Recommended Mitigation: Remove the following line from the comments.

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

Likelihood & Impact :

- Impact : NONE
- Likelihood : ALWAYS
- Severity : Informational

Summary

Severity	No of issues found
HIGH	2
MEDIUM	0
LOW	0
INFO	1
TOTAL	3