Prepared by: mhmojtaba
Lead security researcher: mhmojtaba

- Audit Date: November 25, 2024

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The `mhmojtaba` team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
src/
--- PasswordStore.sol
```

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Gas Optimizations	0
Total	3

Findings

High

[H-1] storing password on-chain make it visible on everyone, no matter what visibility is set on the varable. that means the private visibility is not actually make the variables private and they can be accessed by anyone. No longer Private

Description: all data which is stored on-chain is visible to everyone and can be read by anyone from blockchain. The `PasswordStore::s_password` variable is intended to be private in the protocol and can be accessed by only the owner of the contract using the function `PasswordStore::getPassword` which intended to be called only by the owner.

Impact: Anyone can read the password which is against the main purpose of the protocol.

Proof of Concept:
the below code snippet shows that the `PasswordStore::s_password` variable is not private and can be accessed by anyone from blockchain:

1. Create a local chain

   ```
   anvil
   ```

2. Deploy the contract

   ```
   make deploy
   ```

3. Run the storage tool
   we use `1` as that's the storage slot of the `PasswordStore::s_password` variable in the contract.

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

   we'll get the output as below:
   `0x6d7950617373776f72640000000000000000000000000000000000000000000014`
   which is the bytes32 of the password `myPassword`

   you can then parse that bytes32 to string :

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

   and get an output of : `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.

[H-2] `PasswordStore::setNewPassword` has no access control. so anyone can set a password or change the password.

Description: the function `PasswordStore::setNewPassword` is not protected against setting password by non-owner and everyone can set a password. And due to natspec of the function , `only owner must be able to set a password`.

```
function setPassword(string memory newPassword) external {
>>>     // @audit -vulnerability- missing access control
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Impact: a non-owner can set a password or change the password easily. which is against the main purpose of the protocol.

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

▶ Code

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

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

Recommended Mitigation: Add an access control modifier/conditional to the function `PasswordStore::setNewPassword`:

```
if(msg.sender != owner) revert PasswordStore__NotOwner();
```

Medium

Low

Informational

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

Description: the natspec of the function `PasswordStore::getPassword` mentioned a parameter `newPassword The new password to set` which is not exist. actually the function doesn't have any parameter as this is a getter function.

```
/*
     * @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 line from the natspec.

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

Gas