PasswordStore Audit Report

Prepared by: Jack Landon

Prepared by: Jack Landon Lead Auditors:

- Jack Landon

Table of Contents

Protocol Summary

PasswordStore is 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.

Disclaimer

Jack Landon 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

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	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 in this dcument correspond to the following commit Hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

- Owner: The user who can set the password and read the password.
- Outsiders: No one else should be able to set or read the password.

Executive Summary

The audit was straightforwad, especially considering the simplicity of the scoped contract.

I spent 2 hours using foundry to conduct the audit.

I found 3 issues; 2 high and 1 informational.

Issues found

Severity	Number of Issues Found
HIGH	2
MEDIUM	0
LOW	0

Severity	Number of Issues Found
INFORMATIONAL	1
TOTAL	3

Findings

High

[H-1] Storing the password on in storage on chain makes it visible to anyone.

Description: All data stored on chain is visibible and can be read by anyone. Storing the password on-chain means that you should never rely on the password being secret. The `PasswordStore::s_password` is intended to be private and intended to be only called by the owner of the password.

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

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

Proof of Concept: The below test case shows how anyone can read the `PasswordStore::s_password` variable off-chain.

1. Start up anvil:

```
anvil
```

2. Deploy the `PasswordStore` contract to the anvil chain:

```
make deploy
```

3. Read the `PasswordStore::s_password` storage slot variable off-chain, where 1 is the `s_password` storage slot:

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

You'll get an output that looks like this:
`0x000000000000000000000000f39fd6e51aad88f6f4ce6ab8827279cfffb92266`

4. The output will return a bytes32 string, so parse the hex like this:

```
cast bytes-32-string
0x000000000000000000000000f39fd6e51aad88f6f4ce6ab8827279cfffb92266
```

You'll get an output that looks like this:

`myPassword`

Which is the password set when on contract deployment.

Recommended Mitigation: Due to the nature of this issue, the whole protocol should be rethought. Perhaps the intended password should be encrypted off-chain and only the hash is stored on-chain, to be decrypted off-chain. This would require the user to remember another password to decrypt the hashed password.

Likelihood & Impact:

- Impact: HIGH - It severely interrupts the protocol functionality
- Likelihood: HIGH - The password can be accessed and decoded off-chain

[H-2] Access control not implemented for `PasswordStore::setPassword`, meaning anyone can change the password.

Description: The `PasswordStore` contract intends for only the owner of the password to be able to set the password. However, the `PasswordStore::setPassword` function has no check for the `msg.sender`, and the `PasswordStore::s_password` state variable is able to be changed by non-owners.

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

Impact: Anyone can change the `PasswordStore::s_password` storage variable, severely breaking the functionality and intention of the contract.

Proof of Concept: The below test case proves in a test form that any random address can set the password, and when the owner calls `PasswordStore::getPassword`, they will get the new password, set by the attacker.

▶ Code

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

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

This test case should be added to the `PasswordStore.t.sol` test file.

Recommended Mitigation: Add an access control check on `msg.sender` to the `PasswordStore::setPassword` function.

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

Likelihood & Impact:

- Impact: HIGH - It severely interrupts the protocol functionality
- Likelihood: HIGH - Anyone can change the password
- Severity: HIGH

Informational

[I-1] Documentation Error: Incorrect NatSpec for `PasswordStore:getPassword` Function

Description: The `PasswordStore::getPassword` function presents as follows:

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @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;
    }
```

In the natspec, it suggests that function signature should be `getPassword(string)`, and the implementation is `getPassword()`.

Impact: There is inconsistency between the natspec and the implementation of the function, potentially causing confusion for developers and users of the contract.

Recommended Mitigation: Remove the incorrect natspec line:

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

Likelihood & Impact:

- Impact: NONE - It does not affect the protocol functionality
- Likelihood: NONE - It is a documentation error
- Severity: INFORMATIONAL