PasswordStore Audit Report

Prepared by: hpsb

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's password. 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 store and use this password.

Disclaimer

I make all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit by me 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

Audit Details

Commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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 went well. This was my first private report and it took me some hours to get here. Overall a fun process. Main tool I used was foundry.

Issues found

Severity	Number of Issues
High	2
Medium	
Low	
Info	1
Gas	
---------	----------------
Total	3

Findings

High

[H-1] Storing The Password On-Chain Makes It Visible To Anyone, And No Longer Private

Description: All the data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordSTore::s_password` variable is intented to be a private variable and only accessed through the `PasswordStore::getPassword` function, which is intented to be only called 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, severly breaking the functionality of the protocol.

Proof of Concept: Step 1: Deploy a local blockchain

```
anvil
```

Step 2: Deploy the contract

```
make deploy
```

Step 3: Get the contract address and RPC-URL to use `cast storage` in the required storage slot to access it

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

Step 4: Convert the given hexadecimal to a string

```
cast parse-bytes32-string
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You get the following output:

```
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::setPassword` is callable by anyone

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`.

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

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

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

▶ Code

```
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 modifier to the `setPassword` function.

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

Informational

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

Description:

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

The natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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