PasswordStore Audit Report

Version 1.0

LipeTuga

January 16, 2024

PasswordStore Audit Report

Filipe Magalhães

January 15, 2024

Prepared by: Filipe Magalhães

Lead Auditors: - Filipe Magalhães

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

Filipe Magalhães has make 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
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 described in this document correspond the following commit hash:

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.

Issues found

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

Findings

High

[H-1] Variables stored in storage on-chain are visible to anyone, and no longer private

Description: All data store 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 `PassowrdStore::getPassword()` function, wich is intended to be only called by the owner of the contract.

Impact: Anyone can read the password stored in the contract, serverly breaking the functionality of the contract.

Proof of Concept: (Proof of code) The below test case shows how anyone can read the password directly from the blockchain, without calling the `getPassword()` function.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract

```
make deploy
```

3. Run the storage tool

We use 1because that's the storage slot of the `s_password` in the contract.

```
cast storage <ADDRESS_HERE> 1 --RPC-URL=http://localhost:8545
```

You get a output that looks like this: '0x6d7950617373776f726400000000000000000000000000000000000000000014"

You can the parse that hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One couuld 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 likey want to remove the view function as you wouldn't want the user to accedentally send a transaction with the password that decrypts your password.

[H-2] The `PasswordStore::setPassword()` has no access controls, meaning a non-owner can set the password

Description: The `PasswordStore::setPassword()` function is set to be an `external` function, however, the natspec of the funciton 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
        s_password = newPassword;
        emit SetNewPassword();
    }
```

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

Proof of Concept: Add the following to the `test/PasswordStore.t.sol` 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);
    }
```

***Recomended mitigation:** Add an access control coditional to the setPassword function.

```
    function setPassword(string memory newPassword) external {
        require(msg.sender == owner, "Only the owner can set the
            password");
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Informational

[I-1] The `PasswordStore::getPassword()` natspec indicates a parameter, but the function doesn't take any parameters

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

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

Impact: The natspec is incorrect, and could lead to confusion.

Recomended mitigation: Update the natspec to match the function signature.

```
1      /*
2       * @notice This allows only the owner to retrieve the password.
3  -    * @param newPassword The new password to set.
4       */
```