PasswordStore Audit Report

Version 1.0

Cyfrin.io

May 29, 2024

PasswordStore Audit Report

Mohammad Norouzi

May 29, 2024

Prepared by: Cyfrin Lead Security researcher:

- Mohammad norouzi

Table of Contents

Protocol Summary

The PasswordStore Protocol is for saving passwords on-chain. The password should only be changeable and visible to the owner, not to any non-owner.

Disclaimer

The mohammad norouzi 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:

```
- Commit Hash:  7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

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

Executive Summary

One auditor spent 5 hours auditing to find any possible bugs. ## Issues found

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

Findings

High

[H-1] Variables which stores in storage are public , any one can see the password

Description: Variables which stores in storage are public , the `PasswordStore::s_password` variable is tend to be used only visable from the `PasswordStore::getPassword` function which can only call by the owner of the password

Impact: The `PasswordStore::s_password` variable to anyone

Proof of Concept: (Proof of Code) below test shows how anyone can access to the password

1. Create a locally running chain

```
make anvil
```

2. Deploy the Contract on the chain

```
make deploy
```

3. Read the contract storage a slot 1 because `PasswordStore::s_password` variable located in there

```
cast storage "contract-address" 1
```

You will get some output like this: "0x6d7950617373776f726400000000000000000000000000000000000000000014"

4. Pasre it from bytes to string

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

Now you get the password "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 on-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] The `PasswordStore::setPassword` function can call by even a non-owner, which make the passwored changable by anyone

Description: set the password can be do only by owner while The `PasswordStore::setPassword` function has no any restriction to prvent other from chaining the password

```
    function setPassword(string memory newPassword) external {
@>      // thres no any access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password by calling the `PasswordStore::setPassword` function

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

Code

```

function test_non_owner_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: To check who is calling this function and prevent non-owners from changing the password only needs to add a contition for checking is msg.sender equal to owner add the following at the beginig of the `PasswordStord::setPassword` function:

```

+     if(msg.sender != owner){
+         revert Error()
+     }
```

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.

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