Protocol Audit Report

Prepared by: Hamid

Table of Contents

Protocol Summary

The password storage protocol 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

The YOUR_NAME_HERE 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

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

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

Audit Details

** The findings described in the document correspond with th following commit hash: **

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

-Owners: The userswho can set the password and retreive the password back. -outsiders: no one else should be able to set the password and retreive the password back

Executive Summary

the audit was fun, actually this is the second time im going through this course again. it took me a long time to understand but im gladd im getting better at it, thanks patrick collins

Issues found

|Severity | | No of issues found | |---------| | -------------------| | High | | 2 | | medium | | 0 | | low | | 0 | | info | | 1 | | total | | 3 |

Findings

High

[H-1] Storing the password on chain makes it visible to everyone and no longer private.

Likelihood & Impact

- IMPACT: HIGH

- LIKELIHOOD: HIGH
- SEVERITY: HIGH

Description: All data stored on-chain is viible to anyone, and can be read directly from the blockchain.the `passwordStore:: s_password` vairable is intented to be a private vaariable and only accessed to be a `passwordStore:: getPassword` function, which is interndend to be only called by the owner of the contract.

Impact: Anyone can read the private password, severly breaking the functionality of the protocol.

Proof of Concept: {Proof of code} The below test shows how anyone can read the private password from the blockchain

1. **run:**

```
anvil
```

2. **deploy the contract to local chain:**

```
make deploy
```

3. **run the storage tool:** I used slot `1` because the s_password storage variable is in slot 1.

```
cast storage "contract addr" 1 --rpc-url http://localhost(anvils in this case)
```

4. **run storage tool to convert bytes to strings:** I Ran and convert the byte code gotten from NO 3, to string in your CLI configuration

```
cast parse-bytes32-string "(bytes code gotten from NO 3)
```

5. **And the storage CLI Output:**

```
myPasswords
```

Recommended Mitigation: Due to the critical findings, it is noted that the architecture to be used in defining the contract should be reconstructed, because the anyone can set a password, the solution to this to make the store the password off chain and encript the password return with a fucntion to store the password on the blockchain. this will require the user to remember another password offchain to decript the password. You would also like 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` Missing Access control, Anyone can store a password and this breaks the poassword logic and code

IMPACT & LIKELYHOOD

- IMPACT: HIGH
- LIKELYHOOD: HIGH -SEVERITY: HIGH

Description: The Aim of the the project is to store password informations, anyone who isn't the initial caller can act as a user can go and store a password by calling the setPassword function and this breaks the poassword logic. The `password: :setpassword` function is set to be an external function however, the natspec of the function and overaall purposes of the start contract is that `this function allow only the owner to set a new password.`

```
        function setPassword(string memory newPassword) external {
@>      // @audit - there is a missing access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set the password, severaly breaking the password logic contract

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

▶ CODE

```
    function test_anyOneCanSetPassword(address randomAddress) public {
        //remeber that this is a fuzz test, and a fuzz test when there are
random (data or uint or string or bytes) thrown at a function
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory expectedPassword = "newPassword";
        passwordStore.setPassword(expectedPassword);
        vm.stopPrank();

        vm.startPrank(owner);
        string memory actualPassword = passwordStore.getPassword();
        vm.stopPrank();

        assertEq(expectedPassword, actualPassword);
    }
```

Recommended Mitigation: The recommended mitigation will be that you add an access control conditionals to the `setPassword` function.

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

```
    }
```

Informational

[I-1] `PasswordStore:: getPassword` natSpec indicated a param that doesn't exist, This causes the netspec to be incorrect.

Description: The `PassowrdStore:: getPassword` function signatue is `getPassword()` which the natspac says it should be `getPassword(string)`.

IMPACT & LIKELIHOOD

- LIKELIHOOD: HIGH
- IMPACT: NONE
- SEVERITY: INFORMATIONAL/ GAS/ NON CRITS

Impact: The natspec is Incorrect

Recommended Mitigation: Remove the natspec line

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