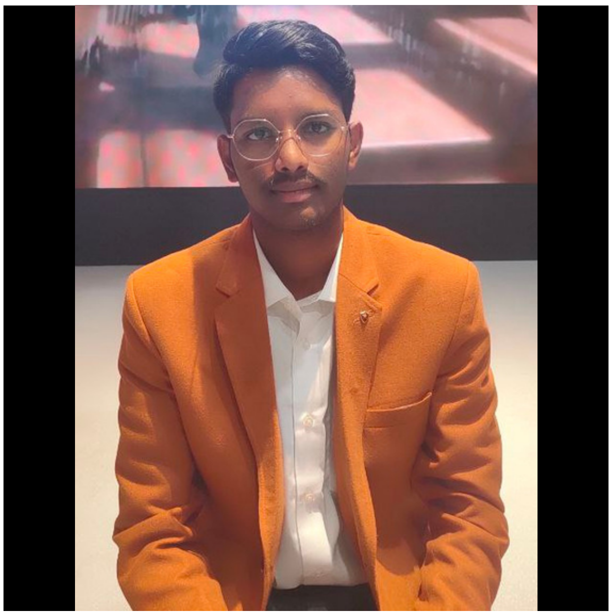

Protocol Audit Report

Version 1.0

Cyfrin.io

December 12, 2023

PasswordStore Audit Report

akhilmanga

December 2, 2023

Prepared by: Akhil Manga

Lead Security Researcher:

- AKHIL MANGA

Table of Contents

- Informational
  * [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect
- Gas
  * [G-1] Reduce the gas price by removing `memory` keyword, it takes more gas price.

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of 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

The AKHIL MANGA 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

Scope

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

Roles

- Owner: The only user, eho can set the password and read the password.
- Outsiders: No one else should be able to setor read the password

Executive Summary

- I'm the only one audited this project and found 2 highs, 1 informational, 1 gas optimization

Issues found

Severity	Number of issues found
High	2
Medium	0
low	0
Informational	1
Gas	1
Total	4

Findings

High

[H-1] Storing the password onchain will be seen by everyone, so the password will not be private

Description: All data stored on-chain will be visible to everyone, and can be read directly from the blockchain. The `PasswordStore::s_password` must be private variable and only accessed through `PasswordStore::getPassword` function, which is only called by the owner.

Impact: Annyone can read the private password. It breaks the functionality of the protocol.

Proof of Concept:

The below test case shows how anyone can read the password directly from the blockchain

1. create a local running chain

```
make anvil
```

2. Deploy contract to the chain

```
make deploy
```

3. Running the storage tool

we use 1 because that's the storage slot `s_password` in the contract

```
cast storage <CONTRACT_ADDRESS> 1 --rpc-url http://127.0.0.1:8845
```

After above command, output looks like this

`0x6d7950617373776f726400000000000000000000000000000000000000000001`

Parse that hex to a string with

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

After the above command, output looks like this:

```
myPassword
```

Recommended Mitigation:

```
-   `string private s_password;`
+   `bytes32 private s_passwordHash;`
```

=> In the `PasswordStore::setPassword` function:

```
-   `s_password = newPassword;`
+   `s_passwordHash = keccak256(abi.enocodePacked(newPassword))`
```

=> In the `PasswordStore::getPassword` function:

```
-   `s_password`
+   `s_passwordHash`
```

=> `passwordStore::getPassword` function looks like:

```
function getPassword() external view returns (bytes32 memory) {
if (msg.sender != s_owner) {
revert PasswordStore__NotOwner();
}
return s_passwordHash;
}
```

[H-2] `PasswowrdStore::setPassword` has no access control, any user can change the password

Description: In the documentation, function allows only owner to set a new password. Because, the absense of the access control anybody can change the password.

```
    function setPassword(string memory newPassword) external {
        // @audit Missing the access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set the new password of the contract. It is breaking the contract's functionality

Proof of Concept: Add the below code 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 access control to the `PassswordStore::setPassword` function

```
if(msg.sender != 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 `PasswordStore::getPassword` function signature is `getPassword()` which the natspec say it should be `getPassowrd(string)`.

Impact: The natspec is incorrect

Proof of Concept: Remove the incorrect natspec line.

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

Recommended Mitigation:

Gas

[G-1] Reduce the gas price by removing `memory` keyword, it takes more gas price.

Description:

```
function setPassword(string memory newPassword) external {
   // @audit-gas remove memory keyword
       s_password = newPassword;
       emit SetNetPassword();
   }
```

Impact:

By removing memory keyword, it reduces the gas price of the contract. Just a minimal gas price drops.

Proof of Concept:

Memory stores the data, but calldata just reads the data. That’s why we use calldata to spend less gas price

Recommended Mitigation:

```
-   function setPassword(string memory newPassword) external {
     }
+   function setPassword(string calldata newPassword) external {
     }
```