Protocol Audit Report

Version 1.0

Cyfrin.io

October 31, 2024

Protocol Audit Report

Vivek Mitra

March 7, 2023

Prepared by: MintXer

Table of Contents

Protocol Summary

Contract's main function is the ability to store passwords that others can't see and allow only owner to update it

Disclaimer

The auditor 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 in this report correspond to the following commit hash:

```
6e2ae63daca11bb6f5220b9fb19619c68748fcef
```

Scope

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

Roles

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

Executive Summary

- Used manual review to analyze the code base
- Audit took 2 hours
- Codebase for simple to review with less than 50 lines of code

Issues found

Severity	Number 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 anyone, and no longer private

Description: All data stored on-chain is visible to anyone and can be directly read from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accessed through the `PasswordStore::getPassword()` function which is intended to be called by the owner of the contract

Impact: Anyone can read the password severely breaking the functionality of the protocol

Proof of Concept: Proof of code The below test case demonstrates how anyone can read from the blockchain

1. Create a local chain using `Anvil` :

```
1      make anvil
```

2. Deploy contract:

```
1 make deploy
```

3. Use `cast` to read storage:

```
1 3-passwordstore-audit % cast storage 0
     x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http://
     127.0.0.1:8545
2 0x6d7950617373776f726400000000000000000000000000000000000000000014
```

> the password is stored in the '1' storage slot

4. Use `cast` to parse the output

```
1 3-passwordstore-audit % cast parse-bytes32-string 0
     x6d7950617373776f726400000000000000000000000000000000000000000014
2 myPassword
```

Recommended Mitigation: Overall architecture of the contract should be rethought. You can store password offchain, and encrypt it and store the encryption on-chain. The user would also need to remember a password off chain to decrypt the main password. Avoid using view functions incase user sends a transaction that exposes the decryption password.

[H-2] `PasswordStore::setPassword()` does not have access control so non-owner can change the password

Description: The `PasswordStore::setPassword()` has function visibility set to `external` however according to the @NatSpec documentation - `@notice This function allows only the owner to set a new password.`

```
1   /*
2       * @notice This function allows only the owner to set a new
            password.
3       * @param newPassword The new password to set.
4       */
5      //@audit anyone can set a password
6      //missing access control
7      function setPassword(string memory newPassword) external {
8          s_password = newPassword;
9          emit SetNetPassword();
10     }
```

Impact: Anyone can set the password, severely breaking the contract's functionality

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

```
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 conditional to the `PasswordStore::setPassword()` function:

```
function setPassword(string memory newPassword) external {
    if(msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Medium

Low

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a parametre that doesn't exist resulting in incorrect natspec

Description: The natspec for the `PasswordStore::getPassword` indicates a parametre to be passed into the function but no parametre is passed into the function:

Code

```

 /*
     * @notice This allows only the owner to retrieve the password.
     * @audit no newPassword parametre passed to the function
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

```
8           if (msg.sender != s_owner) {
9               revert PasswordStore__NotOwner();
10          }
11          return s_password;
12      }
```

Impact: Impacts code readabilility and documentation

Proof of Concept: N/A

Recommended Mitigation: Remove `* @param newPassword The new password to set.` from function natspec

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

Gas