Protocol Audit Report

Version 1.0

https://github.com/PranavRJoshi

April 20, 2024

Protocol Audit Report

Pranav Ram Joshi

April 20, 2024

Prepared by: Pranav Ram Joshi Lead Auditors: - Pranav Ram Joshi

Table of Contents

Protocol Summary

PasswordStore is a protocol used primarily for storing and retrieving user's password. The protocol is designed for single user and not multiple users. Only the owner should be able to set and get the access to password.

Disclaimer

Pranav Ram Joshi and 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 corresponds to the following 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.

Executive Summary

The audit was quick as the source file did not consist of much code. Couple of high severity issues along with an informational issue were found during the audit.

Foundry was used as the test framework.

Issues found

Severity	Number of Issues Found
High	2
Medium	0
Low	0
Informational	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 read directly 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 only called by the owner of the contract.

We show one such method of reading any data off chain below.

Impact:

Anyone can read the private password, which breaks the actual functionality of the protocol.

Proof of Concept: (Proof of Code)

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

1. Create a locally running chain:

```
make anvil
```

2. Deploy the contract to the chain:

```
make deploy
```

3. Run the storage tool:

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

    ```
    cast storage <ADDRESS-HERE> 1 --rpc-url http://127.0.0.1:8545
    ```

    2. You'll get an output that looks like this:

    `0x6d7950617373776f726400000000000000000000000000000000000000000014`

    3. You can parse the hex value to string using:

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

    4. This returns the value:

    `myPassword`

Recommended Mitigation:

With the nature of the protocol, the overall architecture it should be reconsidered. One could encrypt the password off-chain and then store the encrypted password on-chain. However, this would require the user to remember another password off-chain to decrypt the on-chain encrypted password. Also, you'd also likely want to remove the view function as you wouldn't want the user to accidently send a transaction with the password that decrypts your password.

Password –> Encryption Layer –> Encrypted Password –> Fetch Password –> Decrypt Password –> Original Password

Stage	Chain
Password	Off-Chain
Encryption Layer	Off-Chain
Encrypted Password	On-Chain
Fetch Password	On-Chain
Decrypt Password	Off-Chain
Original Password	Off-Chain

[H-2] `PasswordStore::setPassword` has no access control, meaning a non-owner could change the password

Description:

`PasswordStore:setPassword` function has the external visibility, however the natspec of the function (@notice) 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 SetNetPassword();
    }
```

Impact:

Anyone can set or change the password of the contract, severly breaking the contract's intended functionality.

Proof of Concept:

Add the following code to the `PasswordStore.t.sol`

Code

```
    function test_anyone_can_set_password(address random_address)
        public {
        vm.assume(random_address != owner);
        vm.prank(random_address);
        string memory expected_password = "myNewPassword";
        passwordStore.setPassword(expected_password);

        vm.prank(owner);
        string memory actual_password = passwordStore.getPassword();
        assertEq(actual_password, expected_password);
    }
```

Recommended Mitigation:

Add an access control condition to the `setPassword` function.

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

Medium

Low

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
    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()` while the natspec says it should be `getPassword(string)`.

Impact:

The natspec is incorrect.

Recommended Mitigation:

Remove the incorrect natspec line.

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

Gas