Protocol Audit Report

Version 1.0

Kotte Mohan krishna

April 20, 2025

Protocol Audit Report

Kotte Mohan Krishna

April, 2025

Prepared by: Mohan Kotte Lead Security Researcher: Mohan Kotte

Table of Contents

Protocol Summary

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 Mohan Kotte 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:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

- Owner: The user who can set the password and read the password
- Outsiders: No one else should be ab;e to set or read the password

Executive Summary

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 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 owner of the contract

we show one such method of reading any private data on-chain below

Impact: Anyone can read the private password, which breaks the functioanlity of the protocol

Proof of Concept:(Proof of code)

The below testcase 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

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

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

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get an output of:

```
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 off-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] `PasswordStore:setPassword` has no access control missing, meaning a non-owner could change the password

Description: `PasswordStore:setPassword` is a external functions which can be called by anyone, but the protocol is intended to `This function can be only access by Owner of contract 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/change the password of the contract, severely breaking the contract intended functionality

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

Code

```
1
2     function test_anyone_can_set_password(address randomAddress) public
          {
3         vm.assume(randomAddress!=owner);
4         vm.prank(randomAddress);
5         string memory expectedPassword = "myNewPassword";
6         passwordStore.setPassword(expectedPassword);
7
8         vm.prank(owner);
9         string memory actualPassword = passwordStore.getPassword();
10        assertEq(actualPassword, expectedPassword);
11    }
```

Recommended Mitigation: Add an access control condition to the `setPassword` function

```
1           if (msg.sender != s_owner) {
2             revert PasswordStore__NotOwner();
3         }
```

Informational

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

Description:

```
1      /*
2      * @notice This allows only the owner to retrieve the password.
3      * @param newPassword The new password to set.
4      */
5     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

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