PasswordStore Audit Report

Rahber Ahmed

March 22,2025

PasswordStore Audit Report

Version 1.0

March 23, 2025

PasswordStore Audit Report

Rahber Ahmed

March 22,2025

Prepared by: Rahber Ahmed

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 Rahbar Ahmed 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

– Commit Hash: 7d55682ddc4301a7b13ae9413095feffd9924566

Scope

```
./src/
#── PasswordStore.sol
```

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

Roles

- Owner: The user who can set the password and read the password.
- Outsides: No one else should be able to set or read the password. # Executive Summary I have audited and found some bugs in this protocol which have been submitted here. ## Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Gas	0
Total	3

Findings

High

[H-1] Storing the password on-chain are visible to anyone making the passwords not private.

Description: All data stored on-chain is visible to anyone irrespective of public/private/internal/external modifiers/keywords here Passwordstore::s_password storage variable is stored on chain which makes it available to be read by anyone directly from blockchain,this password though only expected to be read from Passwordstore::getPassword function which is only supposed to be called by the owner,but clearly it is not the case here.

we show such an example of reading data of chain below

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

Proof of Concept: Below test shows how one can read the data(password) from the blockchain

1. Local chain

```
make anvil
```

2. Deploy contract Passwordstore on the local chain

```
make deploy
```

3. Access storage variable passwordstore::s_password which is at slot 1

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

this give output as 0x6d7950617373776f7264000000000000000000000000000000000000000000014
convert it into string

```
cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000
```

this gives us the password stored on chain which is myPassword

Recommended Mitigation: Due to this the whole architecture of protocol need to be rethought.One could be encrypt the password off-chain and store the encrpted 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 view function as you would'nt want user to accidentally send transaction decrypt yur password.

[H-2] `Passwordstore::setPassword` has no access control, meaning anyone can set the password

Description: As Passwordstore::setPassword has no access control which means any user can set password,However this function only designed to allow owner to set the new password.

```
  function setPassword(string memory newPassword) external {

@>      @audit there is no access control
          s_password = newPassword;
          emit SetNewPassword();
    }
```

Impact: Anyone can change the password,severly breaking the functionality of the contract.

Proof of Concept: Add the following test to Passwordstore.t.sol

code

```
        function test_anyone_can_set_password(address randomUser) public{
        vm.assume(randomUser !=owner);
        vm.prank(randomUser);
        string memory expectedPassword="Ahmed";
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword=passwordStore.getPassword();
        assertEq(expectedPassword,actualPassword);

    }
```

Recommended Mitigation: Add access control to the Passwordstore::setPassword

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

[I-1] `Passwordstore` netspec indicates a parameter that does'nt exit

Description:

```
        @notice new password to set
        function getPassword() external view returns(string memory){

        }
The `Passwordstore::getpassword` function signature is `getPassword()` but the n
```

Impact: The netspec is incorrect.

Recommended Mitigation: Remove the incorrect netspec line

```
-        * @notice new password to set
```