PasswordStore Audit Report

Version 1.0

NaviWeb3Audits

April 6, 2024

PasswordStore Audit Report

NaviWeb3Audits

April 06, 2024

Prepared by: NaviWeb3Audits Lead Auditors:

- Navaneethan

Table of Contents

Protocol Summary

A smart contract applicatoin 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 NaviWeb3Audits 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 the document corresponding the fllwing commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

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

NaviWeb3Audits Team reviewed each line d code in the scope and dedicatedly Working to Safeguard your Contract and Protocol

We Spent 10 hours with 3 Auditors using foundrytoolkit to review your Contract

Issues found

Severity	No of issues found
High	2
Medium	0
Low	0
Informational	1
Total	3

Findings

High

[H-1] Storing Password onchain makes it Visible to Everyone and its not actually private

Description all data stored in onchain is visible to everyone and that can be read directly from the blockchain. the `PasswordStore::s_password` is declared as private and can call its value only from `PasswordStore::getPassword()` method. but we can read their values from the blockchain itself.

Impact Anyone can read the Private Password, Severly breaking the functionality of the protocol

Proof of Concepts The Below Test case Shows How Anyone can read the password.

1. create a locally running Chain

```
make anvil
```

2. Deploy the Contract To The Chain

```
make deploy
```

3. run the storage tool

```
cast storage
0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http
    ://127.0.0.1:8545
```

you will get Binary Output Like This

```
0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. you can convert binary to string using this

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

which gives the Real Private Password

```
myPassword
```

Recommended mitigation Due to this overall Architecture is Failing We Cannot store any type of private data onchain because any data in blockchain is visible to everyone. So one could encrypt the password off-chain and store the encrypted password on-chain eventhough the user must remember the password used to encrypt and decrypt. and you must remove the view function becuase if the user accidentally send the password with the transaction then the real password can be decrypted.

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

Description The `PasswordStore::setPassword` function is set to be an `external` function, however the natspec of the function and the overall purpose of the Smart Contract is that `This function allows only owner of the contract can update password`

```
    function setPassword(string memory newPassword) external {
        // There is no access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact Anyone can Set/Update/Change the password of the contract, Severly breaking the contract intented function

Proof of Concepts Add the following test function in `PasswordStore.t.sol` test file.

Code

```

    function test_anyone_can_change_password(address randomAddress)
        public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = 'NaviNewPassword';
        passwordStore.setPassword(expectedPassword);

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

Recommended mitigation Add Access Control to the function

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

Informational

[I-1] The `PasswordStore::getPassword()` natspec indicates a parameter That Doesn't exist, Causing natspec to be incorrect

Description The `PasswordStore::getPassword` signature is `getPassword()` but natspec says it should be like this `getPassword(string)`

Impact The natspec is incorrect.

Recommended mitigation

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