

# **Protocol Audit Report**

Version 1.0

Protocol Audit Report December 22, 2023

# **Protocol Audit Report**

PsychoPunkSage

December 22, 2023

Prepared by: PsychoPunkSage

#### **Table of Contents**

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- · Findings
- High
  - [H-1] Anything stored on-chain can be accessed by anyone, solidity access control (i.e. private, public etc) is only applicable for contracts.
    - \* Description:
    - \* Impact:
    - \* Proof of Concept:
    - \* Recommended Mitigation:

- [H-2] PasswordStore::setpassword() don't have any "Access Control", so even a "non-owner" can change/set the password
  - \* Description:
  - \* Impact:
  - \* Proof of Concept:
  - \* Recommended Mitigation:
- Informational
  - [I-1] PasswordStore::getPassword() doesn't use any parameters, but the documentation mentions about newPassword
    - \* Description:
    - \* Impact:
    - \* Recommended Mitigation:

# **Protocol Summary**

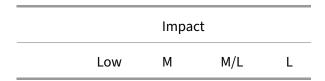
PasswordStore is a protocol that focuses on storage and retrieval of user's passwords. The protocol is meant to be used by single user only. It only allows owners to set and retrieve password.

### **Disclaimer**

I 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   | Н      | H/M    | М   |
| Likelihood | Medium | H/M    | М      | M/L |



We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

### **Audit Details**

The findings described below in this doc is base on following commit hash: 7d55682ddc4301a7b13ae9413099

### Scope

```
1 ./src/
2 |__ 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**

#### **Issues found**

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

## **Findings**

## High

[H-1] Anything stored on-chain can be accessed by anyone, solidity access control (i.e. private, public etc) is only applicable for contracts.

#### **Description:**

PasswordStore::s\_password is accessible to anyone which defeats the protocol ideal This contract allows you to store a **private** password that others won't be able to see. Solidity keywords is only applicable on contracts. But you are storing s\_password on-chain, so, anyone can see it.

#### **Impact:**

**s\_password** is no more safe or private.

#### **Proof of Concept:**

Here is how one can attack your system.

1. Run anvil:

```
1 anvil
```

2. Deploy the Contract

```
1 make deploy
```

3. Read the Storage slot of **s\_password** (i.e. 1) using cast

```
1 cast storage <CONTRACT_ADDRESS> 1
```

4. Decode the data obtained from step:3

Output (on success): 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() don't have any "Access Control", so even a "non-owner" can change/set the password

#### **Description:**

According to natspec of **PasswordStore::setpassword()** i.e. @notice This function allows only the owner to set a **new** password. but the function don't have any access restriction/control. So, anyone can call this function and Change Password. this defeats the intention of the protocol.

#### **Impact:**

Anyone can change the **s\_password**.

#### **Proof of Concept:**

Please paste Test Code attached below to test/PasswordStore.t.sol for checking....

#### Test Code

```
function test_nan_owner_can_set_password(address randomAddress) public
{
    vm.prank(owner);
    string memory owner_pass = passwordStore.getPassword();

    string memory hackedPassword = "HackedPassword";
    vm.prank(randomAddress);
    passwordStore.setPassword(hackedPassword);
```

```
8
9
           vm.prank(owner);
10
           string memory owner_pass_now = passwordStore.getPassword();
11
12
           // To prove:: owner_pass_now != Password set by owner (i.e.
               owner_pass) + owner_pass_now == HackedPassword
13
           assert(keccak256(abi.encodePacked(owner_pass)) != keccak256(abi
               .encodePacked(owner_pass_now)));
           assert(keccak256(abi.encodePacked(hackedPassword)) == keccak256
14
               (abi.encodePacked(owner_pass_now)));
15
       }
```

#### **Recommended Mitigation:**

You can add following Lines of code to PasswordStore::setpassword()

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

#### **Informational**

# [I-1] PasswordStore::getPassword() doesn't use any parameters, but the documentation mentions about newPassword

#### **Description:**

Since PasswordStore: getPassword() don't require any parameters, but the documentation mentions about a parameter newPassword. This suggests the signature of this function is getPassword(string) which is not true.

```
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) {
```

#### Impact:

Can be misleading for future reference.

# **Recommended Mitigation:**

Remove the line from *natspec*:

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