# Protocol Audit Report

Version 1.0

*SachetDhanuka*

June 7, 2024

# Protocol Audit Report

Sachet Dhanuka

June 6, 2024

Prepared by: Sachet Dhanuka Lead Security Researcher - Sachet Dhanuka

## Table of Contents

## Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able set and access the password.

## Disclaimer

Sachet 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

### Scope

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

### Roles

- Owner : Only the owner can set and read the password.
- Outsider : No other user should be able to set and 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 onchain makes it visible to anyone, and no longer private

**Description:** All the data stored on the 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, severly breaking the functionality of the protocol.

**Proof of Concept:** The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

Create a locally running chain

```
make anvil
```

Deploy the contract to the chain

```
make deploy
```

Run the storage tool

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

We use 1 because that's the storage slot of `PasswordStore::s_password`.

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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

### [H-2] The `PasswordStrore::setPassword` has no access controls, meaning a non-owner can change the password

**Description:** The `PasswordStore::setPassword()` 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 the owner to set a new password`.

```
function setPassword(string memory newPassword) external {
    //@audit Access control issue
        s_password = newPassword;
        emit SetNetPassword();
    }
```

**Impact:**

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

Code

```
function test_NonOwnerCanChangeThePassword(address randomAddress)public
    {
        vm.assume(randomeAddress != owner);
        string memory changePassword = "Hacked";
        vm.prank(randomAddress);
        PasswordStore.setPassword(changePassword);

        vm.prank(owner);
        string memory newPassword = PasswordStore.getPassword();
        console.log(changePassword);
        console.log(newPassword);
        assertEq(newPassword,changePassword);
    }
```

**Recommended Mitigation:** Add an access control conditional to the `setPassword()` function.

```
if(owner!=msg.sender){
    revert PasswordStore_NotOwner();
}
```

### Informational

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

**Description:**

```
/*
 * @notice This allows only the owner to retrieve the password.
@> * @param newPassword The new password to set.
 */
function getPassword() external view returns (string memory) {}
```

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.

```
    /*
     * @notice This allows only the owner to retrieve the password.
-    * @param newPassword The new password to set.
     */
```