# PasswordStore Security Review

Version 1.0

*L0W3*

August 15, 2024

## Table of Contents

## Protocol Summary

The protocol audited was an implementation of a password vault, where, acording to the documentation, only the owner should have privileges to read and store passwords into it.

## Risk Classification

| | | **Impact** | | |
|---|---|---|---|---|
| | | High | Medium | Low |
| | High | H | H/M | M |
| **Likelihood** | Medium | H/M | M | M/L |
| | Low | M | M/L | L |

The CodeHawks severity matrix was used to determine severity. See the documentation for more details.

## Audit Details

### Scope

For this review, a Github repository was given: https://github.com/Cyfrin/3-passwordstore-audit/tree/onboarded. The scope of the audit was only set to `PasswordStore.sol` Smart Contract, and therefore, that has been the only contract reviewed. The commit hash where all the tests have been performed is:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

### Roles

As indicated by the documentation provided, the roles given were:

- 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

In this code review, the risk is considered **HIGH** as vulnearbilities with this severety were found, leading to protocol funcrionality break allowing non-owners to set and read passwords.

## Findings

### High

#### [H-1] Variable Stored on Chain Visible to Anyone

**Description**

> The reserved word private in Solidity does not mean that the variable is going to be unreadable, but that it will not be able to be called by other contracts. This data is stored in clear text on the blockchain and it is readable to anyone. `PasswordStore.sol::s_password`

**Impact**

> Publicly stored data in storage variables can lead to secrets disclosure, efectively breaking the funcionallity of this app.

**PoC**

> A Proof of Concept was made, showcasing how the vulnerability could be exploited. That was done by creating a local chain and deploying into ther the contract. Then, the storage of the contract was read, proving that the password was readable. Local Chain

```
anvil
```

Deploy the contract

```
make deploy
```

Read the storage slot of the password

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
    localhost:8545
0x6d7950617373776f7264000000000000000000000000000000000000000014

cast parse-bytes32-string 0
    x6d7950617373776f7264000000000000000000000000000000000000000014
myPassword
```

**Recomendation** It is recommended that the architecture of the app is revised, implementing some type of off-chain encryption.

### [H-2] Insufficient Access Control

#### Description

> As the documentaton says: "@notice This function allows only the owner to set a new password" the password should be only allowed to be set by the owner, however, no check is implemented on the code.

```
function setPassword(string memory newPassword) external { // @audit !!
    This is NOT implementing any access control for owner -> Not owner
    can set passwords
    s_password = newPassword;
    emit SetNetPassword();
}
```

#### Impact

> Allows anyone to set or modify the password, beraking the functionality of the app

#### PoC

> A test calling the funcrion was made and indeed, the modification could be made

```
function test_non_owner_sets_password(address randomAddress) public {
    vm.prank(randomAddress);
    string memory expected = "newPass";
    passwordStore.setPassword(expected);

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

#### Recomendation

> Implement access control features