
# PasswordStore Audit Report

Prepared by: Crypto Auntie

Volume 1.0

# Table of Contents



# Protocol Summary

PasswordStore is a protocol that is dedicated to the storage and retrieval of a user's passwords. Only the owner should be able to set, retrieve, and change the password. This protocol is is designed to be used by one user.

# Disclaimer

The Skywood Ventures 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 |

| | Impact | | | |
|---|---|---|---|---|
| | Low | M | M/L | L |

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

# Audit Details

**The findings described in this document correspond with the follwing commit hash:**

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

## Scope

```
./src/
└── PasswordStore.sol
```

## Roles

- Owner: The user who can set the password and read the password.
- Outsiders: 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] Storing the password on-chain makes it visible to anyone, and no longer private.

**Description:** All data stored on-chain is visible o 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 only be called by the owner of the contract.

One such method of reading any data off chain can be found below.

**Impact:** Anyone can read the private password, severely breaking the functionality of the protocol.

**Proof of Concept:** (Proof of Code)

The test case below shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool We use `1` because that's the storage slot of `s_password` in the contract.

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

You will get an output that look like this:
`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get an output of:

```
myPassword
```

**Recommended Mitigation:** Due to this, the overall structure of the contract should be reevaluated. An option could be to encrypt the password off-chain, then store the encrypted password on-chain. This would require the user to remember another password to decrypt the password. It is also recommended to remove the view function to avoid the user accidentally sending the password that decrypts the password in a transaction.

[H-2] The `PasswordStore::setPassword` function has no access controls, so a non-owner can change the password.

**Description:** The function `PasswordStore::setPassword` has external visibility and does not have an a revert capability if the function is called by a non-owner. Thus, anyone has the ability to set or change the password, which is intention is that `This function allows only the owner to set a new password.`

```
    function setPassword(string memory newPassword) external {
@>      // @audit - There are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

**Impact:** Anyone can set or change the password, severely breaking the contract's intended functionality.

**Proof of Concept:** (Proof of Code)

Add the following to the `PasswordStore.t.sol` test file.

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

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

**Recommended Mitigation:** Add an access control condition to the `setPassword` function.

```
if(msg.sender != s_owner){
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.
@>   // @audit - there is no newPassword parameter
     * @param newPassword The new password to set.
     */

    function getPassword() external view returns (string memory) {
```

The natspec for the function `PasswordStore::getPassword` indicates it should have parameter with the signature `getPassword(string)`. However, the written function signaure is `getPassword()`.

**Impact:** The natspec is incorrect.

**Recommended Mitigation:** Remove the incorrect natspec line.

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