# Protocol Audit Report

Version 1.0

*Cyfrin.io*

November 14, 2024

# Protocol Audit Report

Sauron

August 23, 2024

Prepared by: Sauron Lead Auditors: - Sauron

## Table of Contents

## Protocol Summary

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

## Disclaimer

The Sauron 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 is this document correspond the following commit hash:**

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

**Scope**

```
./src/
|__ PasswordStore.sol
```

### Roles

- Owners: 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

This was my first report as a security reviewer, and I had a bit of trouble understanding it at first. This one took me 5 hours.

### Issues found

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

## Findings

### High

#### [H-1] Storing the password on-chain makes it visable to anyone, and no longer private

**Description:** All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` 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 functionnality 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.

```
1 Create a locally running chain
```

make anvil

```
1 Deploy the contract to the chain
```

make deploy

```
1 Run the storage tool
```

We use 1 because that's the storage slot of s_password in the contract.

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

You'll get an output that looks like this:

```
1 0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

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

And get an output of:

```
1 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.

### Likelihood & Impact

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

### [H-2] `PasswordStore::setPassword` has no access controls, 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 overall purpose of the smart contract is that `This function allow only the owner to set a new password`

```
1     function setPassword(string memory newPassword) external {
2         // @audit - There're no access controls
3         s_password = newPassword;
4         emit SetNetPassword();
5     }
```

**Impact:** Anyone can set/change the password of the contract, severly breaking the contract intended functionality.

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

Code

```
1     function test_anyone_can_set_password(address randomAddress) public
          {
2         vm.assume(randomAddress != owner);
3         vm.prank(randomAddress);
4         string memory expectedPassword = "myNewPassword";
5         passwordStore.setPassword(expectedPassword);
6
7         vm.prank(owner);
8         string memory actualPassword = passwordStore.getPassword();
9         assertEq(actualPassword, expectedPassword);
10    }
```

**Recommended Mitigation:** Add and access control conditional to the `setPassword`function.

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

### Likelihood & Impact

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

## Informational

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

**Description:**

```
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) {
```

The `PasswordStore::getPassword`function signature is `getPassword()` which the natspec says it should be `getPassword(string)`.

**Impact:** The natspec is incorrect.

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

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

## Likelihood & Impact

- Impact: NONE
- Likelihood: NONE
- Severity: Information/Gas/Non-crits