# Protocol Audit Report

Version 1.0

*Locksmith*

March 12, 2024

# Protocol Audit Report

Locksmith

March 12, 2024

Prepared by: Locksmith Lead ASEcurity Researcher: - N.B -Locksmith-

## 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 single 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 Locksmith 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

Commit Hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

### Scope

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

### Roles

- Owner: Is the only one who should be able to set and access the password.
- Outsiders: No one else should be able to set or read the password.

## Executive Summary

### Issues found

| sevevity | 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 visiable to anyone and no longer private (Root Cause + Impact)

-impact: high -likelihood high -severity high

**Description:** all data store on chain is visible to anyone, and can be read directly from the blockchain. the `Passwordstore:s_password` variable is intened to be a private variable and only accessed though 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:** proof of code the below test can 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 <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000000`
 You can then parse that hex to a string with:

```
cast parse-bytes32-string 0
    x6d7950617373776f7264000000000000000000000000000000000000000000014
```

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] `Passwordstore::setPassword` has no access control meaning a non owner could change the password (Root Cause + Impact)

-impact: high -likelihood high -severity high

**Description:** The `Passwordstore::setPassword` function is set to be an `external` function. however, the natspec of function and overall purpose of the smartcontract is `this function allows only the owner to set a new password.`

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

**Impact:** anyone can set/change the password of the contract, breaking the contract intended functionality

**Proof of Concept:** add the following PasswordStore.t.sol test file:

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

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

**Recommended Mitigation:** add an access controol conditional to the `setPassword` function.

"'js 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.(Root Cause + Impact)

-Impact: HIGH likelihood: NONE Serverity: Informational/Gas/non-crits

**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

**Proof of Concept:**

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

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