# Protocol Audit Report

Version 1.0

*[Yusuphoo](https://github.com/yusuphoo)*

January 1, 2025

# Protocol Audit Report

Yusuf Musa

January 1, 2025

Prepared by: Yusuf Musa Lead Security researcher: Yusuf Musa

## Table of Contents

## Protocol Summary

Password is a protocol dedicated for storing and retrieving user's password. the protocol is designed to be used by single User. Only owner should be able to store a password and then retrieve it later. Others should not be able to access the password.

## Disclaimer

The Yusuf's 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 in this document correspond to This Commit Hash:**

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

### Scope

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

### Roles

Owner: The owner who can set the password and read the password. Outsiders: No one else can set the password and read the password.

## Executive Summary

I spent 48 hours Auditing this protocol, using cloc tool and find Three vunnarabilities.

### Issues found

| severity | Number of issues found |
|---|---|
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Info | 1 |
| Total | 3 |

## Findings

### High

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

**Description:** All data stored on chain is public and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function.

**Impact:** Anyone is able to read the private password, severly breaking the functionality of the protocol.

**Proof of Concept:** (proof of code) The below test case shows how anyone can view password directly from the blockchain.

1. Create a locally running chain.

```
make anvil
```

2. Deploy the contract.

```
make deploy
```

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

```
 cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
    http://127.0.0.1:8545
```

You'll get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. 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] `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 purpose of the smart contract and function's natspec indicate 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 SetNewPassword();
}
```

**Impact:** Anyone can set/change the stored password, severly breaking the contract's intended functionality

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

Code

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

        vm.prank(owner);
        string memory acctualPassword = passwordStore.getPassword();
        assertEq(expectedPassword, acctualPassword);


    }
```

**Recommended Mitigation:** Add 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.
@> * @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:** Incorrect Natspec.

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

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