Protocol Audit Report

Version 1.0

Ifra Muazzam

May 16, 2025

Protocol Audit Report

Ifra Muazzam

May 16, 2025

Prepared by: Ifra Muazzam Lead Auditors: - Ifra Muazzam

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and rerieval 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

Ifra Muazzam 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 the following hommit hash: ”’ 7d55682ddc4301a7b13ae9413095feffd9924566 ”’

Scope

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

Roles

- Owner: The user who can set the password and read the password
- Outsiders: No one should be able to 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

Description: All data stored on-chain can be directly read from the blockchian. The `PasswordStore::s_password` variable is intended to be a private variable and only acced through the `PasswordStore::getPassword` function, which is intended to only be called by the owner of the contract.

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

Proof of Concept:

The below test shows how anyone can read from the blockchain

1. Create a locally running chain

```
make anvil
```

2. Deploy Contract to the chain

```
make deploy
```

3. Run the storage tool

we use `1` as 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 a bytes output, something like: 0x6d7950617373776f726400000000000000000000000000000000000000000014

You can then convert that hex to a string with:

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

Recommended Mitigation: Due to this issue, the overall architecture of the contract should be rethought. One could encrypt the password off-chain and then store the encrypted password on-chian. This would require the user to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidently send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no access control, meaning a non-owner could change the password

Description: The `PasswordStore::setPassword` is intended to only be called by the owner of the contract. However there are no access controls to make sure that only the owner can read it.

```
    function setPassword(string memory newPassword) external {
@>      // @audit missing accesss control
        s_password = newPassword;
        emit SetNetPassword();
```

Impact: Anyone can set/change the password of the contract, severely breaking the contracts intended functionality

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

Code

```
    function test_non_owner_can_set_password(address random) public {
        vm.assume(random != owner);
        vm.prank(random);
        string memory changedPassword = "ChangedPassword";
        passwordStore.setPassword(changedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();

        assertEq(changedPassword, actualPassword);

    }
```

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

```
    if (msg.sender != s_owner) {
        revert PasswordStore__MustbeOwner();
    }
```

Informational

[I-1] The `PasswordStore::getPassowrd` natspec indicates a parameter that does not exist

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
@>    * @param newPassword The new password to set.
     */
```

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

Impact: The natspec is incorrect

Recommended Mitigation: Remove the Incorrect Natspec line

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