Ramil Must

Protocol Audit Report

Version 1.0

Ramil Mustafin

June 3, 2025

Protocol Audit Report

Ramil Mustafin

June 2, 2025

Prepared by: Ramil Mustafin

Table of Contents

Protocol Summary

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

Disclaimer

Ramil Mustafin 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 corespond the following commit hash: 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 see or read the password.

Executive Summary

Despite the small number of lines of code, serious vulnerabilities have been discovered in it.

No other tools were used to find vulnerabilities other than knowledge of the Solidity language and a security research course from Cyfrin.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Variables stored in storage onchain are not private

Description: All data stored onchain is visible to anyone. The `PassordStore::s_password` is intended to be a private variable.

Impact: Anyone can read the private password

Proof of Concept: The below test case 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_OF_THE_CONTRACT> 1 --rpc-url http
    ://127.0.0.1:8545
```

You will get output like this: 0x6d7950617373776f72640000000000000000000000000000000000000000000014

You can then parse that hex to a string:

```
cast parse-bytes32-string 0
    x6d7950617373776f72640000000000000000000000000000000000000000000014
```

Output will be: "myPassword"

Recommended Mitigation: Unfortunately, this bug comes from the blockchain architecture itself, so there is no way to store the onchain password covertly. The best solution in this case is to avoid storing the onchain password and only store its derivative. It is also worth removing the password view function, because it may cause a user to accidentally send his password in it

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

Description: The `PasswordStore::setPassword` function is set to be `external` function. The natspec of the function and overall purpose of the smart contract is `This function allows only the owner to set a new password`

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

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

```
    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 conditional to the `setPassword` function

```
    if(msg.sender != 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: 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.

'''diff / @notice This allows only the owner to retrieve the password. - * @param newPassword The new password to set. */ '''