Protocol Audit Report

Version 1.0

https://github.com/tohidul3417

July 5, 2025

Protocol Audit Report

Mohammad Tohidul Hasan

July 4, 2025

Prepared by: Mohammad Tohidul Hasan Lead Security Researcher: - Mohammad Tohidul Hasan

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
  - [H-1] Storing the password on-chain makes it visible to anyone, and no longer private.
  - [H-2] `PasswordStore::setPassword` is missing access control meaning anyone can set the password calling the function.
- Informational
  - [I-1] The `PasswordStore::getPassword` natspec indicates a param which does not exist, casing the natspec to be incorrect.

Protocol Summary

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

Disclaimer

The MTH 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
Likelihood	High	H	H/M	M
	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 describes in this document corresponds to the follwoing commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/PasswordStore.sol
```

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum ## Roles
- Owner: The only user who should be set and retrieve the password
- Outsiders: Nobody except the owner should be able to set or 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, and no longer private.

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

We show one such method reading data off chain below.

Impact: Anyone can read the private password. Severely breaking the functionality of the protocol.
Proof of Concept: The below test case shows how anyone can read the storage variable 1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

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

You will get an output like this `0x64756d6d795f70617373776f726400000000000000000000000000000000000`

4. Then parse that hex to a string with

```
cast parse-bytes32-string 0
    x64756d6d795f70617373776f726400000000000000000000000000000000001c
```

You will get the output (the password that was set): `dummy password`

Recommended Mitigation: The main vulnerability is that all data on a public blockchain is public, regardless of the private keyword. Storing the plaintext password on-chain allows any observer with access to a node to read the data directly, as demonstrated in the proof-of-concept. This completely breaks the contract's promise of privacy. The correct approach is to never store the secret password on-chain. Instead, the contract should only store a cryptographic commitment, such as a keccak256 hash, of the password. This allows the contract to verify a password attempt by hashing it and comparing it to the stored hash, without ever exposing the secret itself.

[H-2] `PasswordStore::setPassword` is missing access control meaning anyone can set the password calling the function.

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

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

Impact: Anyone can set/change the password in the contract, severly breaking 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 = "MyPasswordForTesting";
        passwordStore.setPassword(expectedPassword);

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

        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: Add an access control conditional to `setPassword` function

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a param which does not exist, casing 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) {
```

`PasswordStore::getPassword` function signature is `getPassword` which the natspec says it should be `getPassword(param)` **Impact:** The natspec is incorrectg

Recommended Mitigation: Remove the incorrect natspec line.

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