Protocol Audit Report

Version 1.0

Harsh Suthar

June 4, 2025

Network Vulnerability Assessment

Harsh Suthar

June 4th, 2025

Prepared by: Harsh Suthar

Table of Contents

    * [I-1] `PasswordStore::getPassword` natspec have parameter but the `getPassword` function dont have any paramters

  - Gas

Protocol Summary

A smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

I make 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
```

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

Roles

Owner: The user who can set and get password Outsiders: No one of the outsiders should be able to get or set the password

Executive Summary

I spent half an hour to audit PasswordStore.sol file, and we found 3 issues in total which breaks the functionality of protocol.

Issues found

Severety	Number of issues found
high	2
medium	0
low	0
informational	1
gas	0

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone, making it no longer only private to owner

Description: Anyone can read the storage variables stored on-chain whether they are marked with `private` keyword or not. The `PasswordStore::s_password` should be private, only readable if the `sender` is `owner` of the contract AND it is getting called from the `PasswordStore::getPassword` function.

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

Proof of Concept:

The below code and guide shows that anyone can read the password even without being the `owner` of contract or without calling the `PasswordStore::getPassword` function. We use foundry's cast tool to read the storage data directly here.

1. Create a locally running anvil chain

```
make anvil
```

2. Deploy the contract

```
make deploy
```

3. Note the from the output above, it should be listed in the top of output like this

```
== Return ==
0: contract PasswordStore <CONTRACT_ADDRESS>

## Setting up 1 EVM.

==========================
```

4. Using the cast command to get the s_password variable from the storage

```
cast storage <CONTRACT_ADDRESS> 1 --rpc-url http://127.0.0.1:8545
```

You will get an output of type bytes32, which is just converted form of your initial password set by your deployer, which should look like this

```
0x6d7950617373776f726400000000000000000000000000000000000000000014;
```

5. Parse the bytes32 password to string

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

then you will get the output as below

```
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` is missing the access control, leading to an issue where non-owner can set the password

Description: `PasswordStore::setPassword` being the external function without any pre-check/auth-control to see if the `sender` is the `owner` or `non-owner` will lead to anyone being able to set the password regardless of them being owner or not. This violates the intended functionality of this protocol i.e `Users should be able to store a password and then retrieve it later. Others should not be able to access the password`

```
    function setPassword(string memory newPassword) external {
@>      // @audit-high No Auth control setupped to check if sender is
    owner or not
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: `non-owner` being able to set the password of the contract

Proof of Concept: Add this following test to your `test/PasswordStore.t.sol` file and run the tests

```
    function test_anyone_can_set_password(address randomAddress) public
        {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory newPassword = "myPassword";
        passwordStore.setPassword(newPassword);
        vm.stopPrank();
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, newPassword);
```

```
10      }
```

The above test passing proves that any random address can set the password even without being the owner.

Recommended Mitigation: Add conditional to check if the `sender` is `owner` or not

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

Medium

Low

Informational

[I-1] `PasswordStore::getPassword` natspec have parameter but the `getPassword` function dont have any paramters

Description:

```
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory)
```

`PasswordStore::getPassword` mentions that it will have a parameter in function but there are no parameter named `newPassword` inside getPassword function

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line:

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

Gas