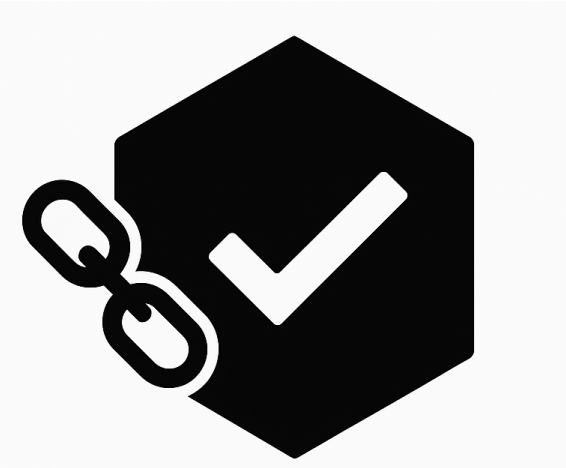

PasswordStore Protocol Audit Report

Version 1.0

github.com/lucasfhope

September 12, 2025

PasswordStore Protocol Audit Report

Lucas Hope

September 12, 2025

Prepared by: Lucas Hope

Table of Contents

Protocol Summary

The protocol intends to store a password on-chain which can only be accessed by the contract owner. The owner should be the only one to be able to set and view the password.

Disclaimer

Lucas Hope 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 report correspond to the following commit hash:

```
be47a4de6d0ccb6621e075caee60409c3f7eac28
```

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 set or read the password.

Executive Summary

The audit found that the design of this protocol is flawed since the password is stored unencrypted on chain. There is also a flaw with how access control is handled.

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 make it visible to anyone and no longer private

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

We show one such method of reading any data off chain below.

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

Proof of Concept: The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
1 make anvil
```

2. Deploy the contract to the local chain

```
make deploy
```

3. Run the cast storage tool

Use the contract address thatw as deployed and use 1 since that is the storage slot of `PasswordStore::s_password`. The result of cast storage will be the data in hexadecimal, so use cast –to-ascii to convert the data to a string.

```
% cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
    http://127.0.0.1:8545 | cast --to-ascii
```

You will get the output…

```
myPassword
```

Recommended Mitigation: The overall architecture of the contract should be rethought. You could encrypt the password off-chain and store the encrypted password on-chain. This would require the user to remember the encryption key to decrypt the password.

[H-2] `PasswordStore::setPassword` has no access control, meaning the password can be changed by anyone

Description: The `PasswordStore::setPassword` function is set as external. However, the natspec of the function and the deocumented purpose of the smart contract says 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 password of the contract, breaking the intended functionality of the contract.

Proof of Concept:

Add the followeing test into `PasswordStore.t.sol`.

```
function test_anyone_can_set_password(address randomAddress) public {
    vm.prank(randomAddress);
```

```
3       string memory expectedPassword = "myNewPassword";
4       passwordStore.setPassword(expectedPassword);
5
6       vm.prank(owner);
7       string memory actualPassword = passwordStore.getPassword();
8       assertEq(actualPassword, expectedPassword);
9   }
```

Then you can run the test, showing that any address can set the password.

```
1 forge test --mt test_anyone_can_set_password
```

Recommended Mitigation: Add an access control conditional at the beginning of the `PasswordStore::setPassword` function.

```
1 if(msg.sender != owner) {
2     revert PasswordStore__NotOwner();
3 }
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist, causing the natspec to be incorrect

Description:

```
1       /*
2       * @notice This allows only the owner to retrieve the password.
3   ->  * @param newPassword The new password to set.
4       */
```

The `PasswordStore::getPassword` function signature is `getPassword()`, while the natspec suggests it is `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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