



# Protocol Audit Report

Version 1.0

*h3xmora*

March 25, 2025

# Protocol Audit Report

H3xmora

March 25, 2025

Prepared by: H3xmora

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    - \* [H-1] Storing a password on-chain makes it visible to anyone, and no longer private
    - \* [H-2] has no access controls, meaning a non-owner could change the password of the owner
  - Informational
    - \* [I-1] the `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing 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 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

h3xmora 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 them 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

I 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 commit hash:**

```
1 7d55682ddc4301a7b13ae9413095feffd9924566
```

## Scope

```
1 .src/  
2 --- 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

### Issues found

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

## Findings

### High

#### [H-1] Storing a 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 the blockchain. The `PasswordStore : : s_password` variable is intended to be a private variable and only accessed through the `PasswordStore : : getPassword` function, 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, severely breaking the functionality of the protocol.

**Proof of Concept:**(Proof of Code)

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 chain

```
1 make deploy
```

- ### 3. Run the storage tool

We use 1 because that's the storage slot of `s_passowrd` in the contract.

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

You'll get an output that looks like this:

[illegible]

You can the parse that hex to a string with:

[illegible]

And get an output of:

```
1 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 password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

**[H-2] has no access controls, meaning a non-owner could change the password of the owner**

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

```
1 function setPassword(string memory newPassword) external onlyOwner
2 {
3     // @audit missing access control
4     s_password = newPassword;
5     emit SetNewPassword();
6 }
```

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

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

Code

```
1 function test_after_access_control() public {
2     vm.startPrank(owner);
3     string memory expectedPassword = "myNewPassword";
4     passwordStore.setPassword(expectedPassword);
5     vm.stopPrank();
6     vm.startPrank(bob);
7     string memory hackedPassword = "hackedPassword";
8     vm.expectRevert();
9     passwordStore.setPassword(hackedPassword);
10 }
```

**Recommended Mitigation:** 1. Add an access control conditional to the `PasswordStore::setPassword` function.

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

2. Add a modifier to the function `PasswordStore::setPassword` and similar functions that require an access control.

```
1 modifier onlyOwner() {
2     require(s_owner == msg.sender, "You're not the owner shoo shoo!");
3     _;
4 }
5
6 function setPassword(string memory newPassword) external onlyOwner{}
```

## Informational

**[I-1] the `PasswordStore::getPassword` natspec indicates a parameter that doesn't 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  */
```

```
5     function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()` which the natspec say it should be `getPassword(string)`.

**Impact:** The natspec is incorrect.

**Recommended Mitigation:** Remove the incorrect natspec line.

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