PasswordStore Protocol Audit Report

Version 1.0

DeniyalDaniDan

April 28, 2025

PasswordStore Protocol Audit Report

deniyaldanidan

April 28, 2025

Prepared by: DeniyalDaniDan

Lead Auditors:

- DeniyalDaniDan

Table of Contents

Protocol Summary

PasswordStore is 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

The **DeniyalDaniDan** 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 described in the document correspond the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Click here to view the Source code

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

Issues found

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

Findings

High

[H-1] Storage the password on-chain makes it visible to anyone, & no longer private

Description: All data stored on-chain ***is public*** and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function.

I'll show one such method of reading any data off chain below.

Impact: Anyone is able to read the private password, severely breaking the functionality of the protocol.

Proof of Concept:

The below test case shows how anyone can read the password directly from the blockchain. We're gonna use foundry's cast tool to read directly from the contract's storage without being its owner

1. Create a local running chain:

```
make anvil
```

2. Deploy the contract on the chain:

```
make deploy
```

3. Run cast:

```
cast storage <CONTRACT-ADDRESS> 1
```

we get an output like:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Convert it to string using cast:

```
cast to-ascii "0
    x6d7950617373776f726400000000000000000000000000000000000000000014"
```

we get the output as `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()` has no access control - anyone can change the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however the purpose of the smart contract and function's natspec indicate 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();
```

```
5 }
```

Impact: Anyone can set/change the password, severly affecting the contract's intended purpose.

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

```
1  function test_anyone_can_set_password(address randomAddress) public {
2      vm.assume(randomAddress != owner);
3
4      // arrange
5      string memory expectedPassword = "password_set_by_non-owner";
6
7      // act
8      vm.prank(randomAddress);
9      passwordStore.setPassword(expectedPassword);
10
11     vm.prank(owner);
12     string memory actualPassword = passwordStore.getPassword();
13
14     // assert
15     assertEq(expectedPassword, actualPassword);
16 }
```

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

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

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()` while the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

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