Protocol Audit Report

Version 1.0

Moon.io

September 2, 2025

Protocol Audit Report

Moon.io

March 7, 2023

Prepared by: Moon Lead Security Researcher:

- Divyansh Audichya

Table of Contents

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

The MOON 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
	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 document correspond the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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'

We used Foundry and found many bugs including the private keyword and access control missing vulnerblities

Issues found

Severitity	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 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 od reading any data off chain below.

Impact: Anyone can read the private password, severly 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

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

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

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

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get an output of:

```
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.

Likelihood & Impact :

- Impact:HIGH
- Likelihood:HIGH
- Severity:HIGH

[H-2] `PasswordStore::setPassword` has no access controls, meaning a anyone who is not an owner can change the password.

Description: The function `PasswordStore::setPassword` don't have access control, which is a critical issue because anyone can change the password ,and hence this will be the failure of this protocol .

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

Impact: Anyone can change the password of the contract, breaking the contract's intended functionality

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

Code

```
    function test_anyone_can_set_the_password(address randomAddress)
        public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory newPasswordByRandomAddress = "
            randomGuyChangedYourPassword";
        passwordStore.setPassword(newPasswordByRandomAddress);

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

        assertEq(actualPassword, newPasswordByRandomAddress);
    }
```

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

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter which doesn't exists, causing the ntspec 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) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

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

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