Password Security Review

Version 1.0

https://github.com/Blockitus

January 1, 2024

Password Security Review

Blockitus

December 01, 2024

Powered by: Cyfrin

Lead Auditors:

- Pedro Machado

Personal youtube channel for web3 spanish speaker developers: Blockitus

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's password. 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 to this password.

Disclaimer

The Blockitus team with its lead auditor: Pedro Machado, 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

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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

This Security Review marks a significant milestone for Blockitus, featuring CEO Pedro Machado as the lead security researcher. Leveraging insights gained from the comprehensive course at Cyfrin, Pedro has honed his skills in Security and Auditing, enriching his knowledge of Web3, Blockchain, EVM, Smart Contracts, and Solidity. With over three years of experience as a blockchain developer, Pedro has contributed to various projects with companies such as Reserva Food System, Fusyona, and Criptoinformativo. Now, he embarks on a new venture with Blockitus, a startup dedicated to crafting impactful DApps that contribute to the evolution of the Web3 market.

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 actually it is visible for everybody.

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

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

Proof of Concept: The bellow process show how anyone can read the password on-chain. We're to cover this using Foundry's `cast` tool for read the data, that should be hidden.

- ☒ **Create a locally running chain** `make anvil`
- ☒ **Deploy Contracts to the chain** `make deploy`
- ☒ **Run the storage tool**

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

- ☒ **You'll get an output that looks like this:**

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

- ☒ **Parse bytes32 to string:**

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

- ☒ **You should retrieved somethiing like**

```
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] The PasswordStore:setPassword function lacks proper access control, anyone can set a password

Description: The protocol intends to set the password just for the smart contract's owner. However, the code `PasswordStore::setPassword(string memory newPassword)` lacks asserting that a non-user set a new password.

Impact: Anyone can set/change the password of the contract, severly breaking the contract intended behavior.

Proof of Concept: We are going through the code bellow that you have to add on `PasswordStore.s.sol` to show how we can exploit this vulnerability.

Code

```
    function test_anyone_can_set_password(address randomAddress) public
        {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "my_new_password";
        passwordStore.setPassword(expectedPassword);
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: Add the line of code that it describes bellow to the top of the function at`PasswordStore::setPassword(string memory newPassword)`.

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec 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)
```

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the natspec wrong line.

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