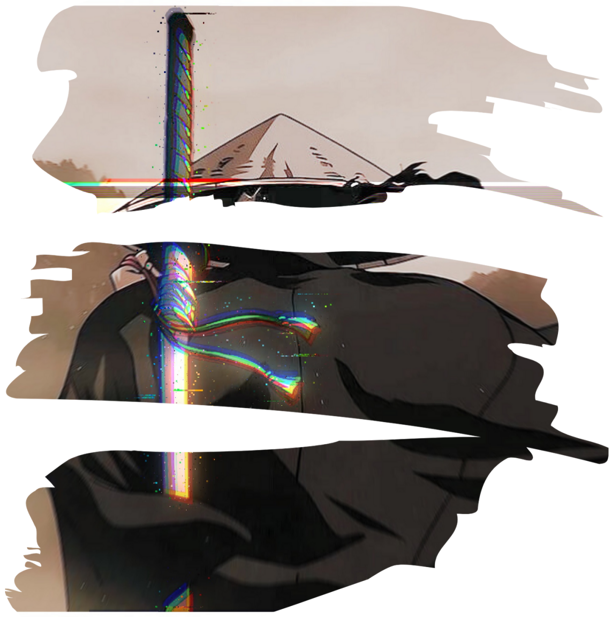

Protocol Audit Report

Version 1.0

0xTekken

January 13, 2024

PasswordStore Audit Report

0xTekken

December 31, 2023

This report was produced by me(0xTekken) during a course by updraft.cyfrin.io: Security & Auditing

Prepared by: 0xTekken Lead Auditors:

- 0xTekken

Table of Contents

Protocol Summary

A smart contract applicatoin 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

0xTekken 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

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

I spent X hours with Z auditors using Y tools… ETC

Issues found

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

Findings

High

[H-1] Storing password on-chain is visible to everyone.

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

Method of reading any data from chain will be shown below.

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

Proof of Concept: (Proof of Code)

The below test showcases how anyone can read the password directly from the blockchain, without the use of `PasswordStore:getPassword` function.

- local PoC:

  1. `anvil`
  2. `make deploy`
  3. `cast storage 0x5fbdb2315678afecb367f032d93f642f64180aa3 1 --rpc-url http://127.0.0.1:8545`
     - `PasswordStore::s_password` is in the storage slot 0x1

  - output: `0x6d7950617373776f7264000000000000000000000000000000000000000000000014`

  - `cast to-ascii 0x6d7950617373776f7264000000000000000000000000000000000000`

    * alternative: `cast parse-bytes32-string 0x6d7950617373776f72640000000000000`

    * output: `myPassword`

- on-chain PoC,

  - Pre-Requisites: add your alchemy/infura/whatever web3 data api you use instead of `--rpc-url`, and the real contract address.

  1. `cast storage 0x2ecf6ad327776bf966893c96efb24c9747f6694b 1 --rpc-url $ALCHEMY_SEPOLIA_API_KEY`
     - output: `0x4c616b6b6164614b616368612100000000000000000000000000000000000`

  2. `cast to-ascii 0x4c616b6b6164614b6163686121000000000000000000000000000000`

     - output: `LakkadaKacha!`

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. Or you would have to have a another set-up for decrypting the stored encrypted password.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password

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

```
    function setPassword(string memory newPassword) external {
@>      /// @custom:issue No Access Control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

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

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

Code

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

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

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

```
        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:

Details

Original natspec:

```
/**
 * @notice This allows only the owner to retrieve the password.
 * @param newPassword The new password to set.
 *
 */
```

Poiting the natspec doc issue:

```
/**
 * @notice This allows only the owner to retrieve the password.
 * param newPassword The new password to set. // @custom:natspec There
    is no parameter for this function.
 *
 */
```

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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