PasswordStore Audit Report

Version 1.0

Bluedragon101

April 4, 2024

PasswordStore Audit Report

Bluedragon101

April 3 2024

Prepared by: Shibi Kishore Lead Auditors: Shibi Kishore

- xxxxxxx

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 Bluedragon101 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 to this document correspond to the following commit hash

`7d55682ddc4301a7b13ae9413095feffd9924566`

Scope

```
src/
--- PasswordStore.sol
```

Executive 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.

Roles

- Owner: Is the only one who should be able to set and access the password.
- Outsides: No one else should be able to set or read the password

For this contract, only the owner should be able to interact with the contract.

Issues found

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

Findings

High

[H-1] Storing the password on-chain it is 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. This includes the password, which is stored in `PasswordStore::s_password`. This is a security risk, as it makes anyone can to get others password using `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 on-chain below.

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

Proof of Concept:

The below test case shows how can anyone read the password stored on-chain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the local 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 <CONTRACT_ADDRESS> 1 --rpc-url http://localhost:8545
```

You'll get a output like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can parse that hex to string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: As this bug changes the whole protocols architecture and functionality, it is recommended to change the way the password is stored. One way to do this is to store a password off-chain, and only use it encrypt/decrypt the on-chain password. This way, the password cannot be decrypted without the offchain password.

[H-2] Missing access control in `PasswordStore::setPassword()`, making non-owner to set a password.

Description: The `PasswordStore::setPassword()` function does not have any access control, allowing anyone to set the password. This is a security risk, as it allows anyone to change the password, which is intended to be only called by the owner of the contract.

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

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

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

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