Protocol Audit Report

Version 1.0

0xbihari

January 30, 2024

Protocol Audit Report

0xbihari

January 30, 2024

Prepared by: 0xbihari

Lead Auditors: - 0xbihari

Table of Contents

Protocol 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.

Disclaimer

The 0xbihari 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 this document correspond to the following commit hash.

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

./src/ #–PasswordStore.sol ## Roles Owner: The user who can set the password and read the password. Outsides: No one else should be able to set or read the password.

Executive Summary

Add some notes about how the audit went, somethings you found, etc. We 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 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` vairable is intended to be a private variable and only accessed by the `PasswordStore::get_password` funtion which is intended to be only called by the owner of the contract.

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

Proof of concept/code: The below test case shows how anyone can read the password directky from the blockchain.

1. Create a locally running chain

```
    make deploy
```

2. 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 now pass that hex to a string value

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

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. Howeve, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with password which decrypts the password.

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

Description: `PasswordStore::setpassword` function is set to be an external function, however, the natspec of the function and the overall purpose of the contract is that `This function allows only owner to set the password.`

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

Impact: Anyone can change/set the password through the contract, `severly breaking the contract functionality`.

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

Code

```
function test_anyone_can_set_password(address randomAddress) external {
        vm.assume(randomAddress != owner);

        vm.prank(randomAddress);
        string memory expectedPassword = "newPassword";
        passwordStore.setPassword(expectedPassword);

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

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

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

Informational

[C-1] The `PassportStore_getPassword()` natspec indicates a parameter indicates a parameter that doesn't exist, making the natspec incorrect.

Description:

```
/*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     * @audit There is no newPassword parameter
     */
$$=> function getPassword() external view returns (string memory) {
```

The `PasswordStore::getpassword()` function signature is `getPassword()` while the natspec says its should be `getPassword(string)`. # Gas