Protocol Audit Report

Prepared by: Daniel Simanullang

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 Daniel Simanullang 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

Scope

7d55682ddc4301a7b13ae9413095feffd9924566

Roles

Daniel Simanullang : Lead Auditor

Issues found

3

Findings

High

[H-1] Variables stored in storage on-chain are 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 of reading any data off-chain below.

Impact:

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

Proof of Concept:

The test case below shows how anyone can read the password directly from the blockchain.

1. Make Anvil

```
make anvil
```

2. Deploy the contract on-chain

```
make deploy
```

3. Cast storage

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
http://127.0.0.1:8545
```

4. Parse the string

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

5. Get the output of

```
myPassword
```

Recommended Mitigation:

Encrypt the password off-chain and store it on-chain. Users need to input the encrypted password in order to decrypt the password.

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

Description:

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

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

Impact:

By it being external, anyone can change the password. Therefore, it breaks the intended functionality of the code.

Proof of Concept:

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

Recommended Mitigation:

Add access control conditional to the `setPassword` function

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

Informational

[I-1] Missing @param `newPassword` in `PasswordStore::getPassword`

Description:

The `getPassword()` function should be `getPassword(string)`.

```
    function getPassword() external view returns (string memory) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

Impact:

Incorrect natspec.

Recommended Mitigation:

Remove incorrect natspec.

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