Protocol Audit Report

Version 1.0

Skipper.io

September 17, 2024

Protocol Audit Report

Sreewin M B

september 17, 2024

Prepared by: [Sreewin m b] Lead Auditors: - 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

Skipper Audits 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

Issues found

severity	Number of issues found
High	2
Medium	0
Low	0
info	1
Total	3

Findings

High

[H-1] Variables stored in storge on-chain are visible to anyone,no matter the solidity visibility keyword.

Description All the 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 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 Concepts(proof of code)

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

1.deploy a locally running anvil chain.

```
make anvil
```

2.deploy the contract on that chain.

```
make deploy
```

3.use this command to read from the storage slot of the deployed contract.

```
cast storage `deployed address` `storage slot` --rpc-url http
    ://127.0.0.1:8545
```

(eg:- cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 –rpc-url http://127.0.0.1:8545)

this returns a bytes32 .To convert a bytes32 to string , we can use

```
cast parse-bytes32-string `returned bytes32 data`
```

we can see the output as `myPassword`.

Recommended mitigation

1.encrypt the password off-chain,and then store the encrypted password on-chain.this would require the user to remember another password off-chainto decrypt the password.However,you'd also likely want to remove the view function as you wouldn't want the user to accidentaly send a transaction with the password that decrypts your password.

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

Description

The `PasswordStore::setPassword` function is set to be an external function,however, the nat-spec of the function and overall purpose of the smart contract 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 Anyone can set/change password ,severly breaking the contracts intended functionality.

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

code

```
function test_anyone_can_set_password(address randomUser) public {
        vm.assume(randomUser != owner);
        vm.prank(randomUser);
        string memory Password = "new Password";
        passwordStore.setPassword(Password);

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

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

```
    /*
     * @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()` while the natspec says it should be `getPassword(string)`

Impact

The natspec is incorrect.

Recommended mitigation

Remove the incorrect natspec line.

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