Protocol Audit Report

Version 1.0

Cyfrin.io

December 12, 2023

PasswordStore Auidt Report

Atulraj Sharma

Decmber 12, 2023

Prepared by: AtulRaj Sharma Private Auditor - xxxxxxx

Table of Contents

Protocol Summary

`PasswordAudit.sol` is a smart contract with a core functionality to store your password securely with no way anyone else retrieving it and only the owner can retrieve or view the password.

Disclaimer

I as an individual auditor made 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/individual 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: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract. # Executive Summary The security review of the `PasswordStore.sol` has been performed by me(Atul), under the guidance of Patrick Collins. ## 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 makes it visible to everyone, an no longer private.

Description: All data stored on-chain is visibile to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accesed through the `PasswordStore::getPassword` function, 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 The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

We use 1 because that is the storage slot of `s_password` in the contract.
```

```
cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You will get an ouput that looks like this: `0x6d7950617373776f726400000000000000000000000000000000`

You can then parse that hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecure of the contract should be rethought. One could encrypt the password off-chain and then upload the encrypted password on-chain. This would require the user to remember another password off-chain.

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

Description: The `PasswordStore::setPassword`function is an external function with no access control and anyone can set or change password. The `PasswordStore::setPassword` is meant to be only accesed by the owner of the contract.

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

Impact: non_owner can change the password, severly breaking the contract's functionality.

Proof of Concept Add the following code to `PasswordStore.t.sol`

Code

```
  function test_non_owner_can_set_password(address randomAddress)
      public {
        vm.startPrank(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 conditional to the `setPassword` function.

```
1
2 if(msg.sender != owner){
3     revert PasswordStore_NotOwner();
4 }
```

Informational

[I-1] `PasswordStore::getPassword` function has an incorrect natspec, no impact

Description: The `PasswordStore::getPassword` has a uselss natspec that states about a parameter in `Password::getPassword` function which is actually not present.

Impact: Incorrect Natspec

Recommended Mitigation: There are no recommended mitgation but this is something you know.