Protocol Audit Report

Version 1.0

Luca3

December 30, 2023

Protocol Audit Report

Luca3

30th December 2023

Prepared by: Luca3 Lead Security Researcher

- Yudhishthra Sugumaran

Table of Contents

Protocol Summary

PasswordStore is 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 Luca3 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 found correspond to this commit hash:

```
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 on how the audit went, types of things found, etc. we spent X hours on the audit with Z auditors using the following tools: X, Y, Z

Issues found

SEVERITY	NUMBER OF ISSUES FOUND
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Password stored in storage is visible to anyone, and not private

Description: All data stored on-chain is visible to anyone. and can be read directly from the blockchain. The `PasswordStore::s_password` varibale is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function, which is supposed to be called by the owner of the contract.

Impact: Anyone can read the private password, severely compromising the security of the contract.

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

```
make deploy
```

3. Run the storage tool 1 is used because the `s_password` variable is stored in that storage slot.

```
cast storage <DEPLOYED_CONTRACT_ADDRESS> 1

# Output
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

After decoding the hex, the password can be seen in plain text.

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 | xxd -r -p

# Output
myPassword
```

Recommended Mitigation:

Storing any crucial information on-chain is not recommended especially if its in plaintext. The intended information should be encrypted with a strong encryption algorithm such as AES-256 alonside a salt value that is randomized and only known to selected users of the protocol. The encrypted data should be stored on-chain and only decrypted off-chain.

[H-2] `PasswordStore::setPassword` function has no access controls, meaning anyone can change the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec 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-info : Missing access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can change the password, severely compromising the security of the contract.

Proof of Concept:

Add the below test case to `test/PasswordStore.t.sol`

Code

```
function test_anyoneCanSetPassword(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.prank(randomAddress);
    string memory fakePassword = "myNewPassword";
    passwordStore.setPassword(fakePassword);

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

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

```
    error PasswordStore__NotOwner();

    modifier onlyOwner {
        if (msg.sender != owner) {
            revert PasswordStore__NotOwner();
        }
        _;
    }

    //existing code

    function setPassword(string memory newPassword) external onlyOwner
       {
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a parameter that does not exist, causing an incorrect natspec.

Description: The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec mentions that it is `getPassword(string)`

Impact: The natspec is incorrect and misleading.

Recommended Mitigation: Remove the incorrect natspec line

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