Protocol Audit Report

Version 1.0

ThangTran

September 5, 2024

PasswordStore Audit Report

ThangTran

Sep 9, 2024

Prepared by: ThangTran Lead Auditors: - ThangTran

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. Only owner should be able to set and access the password.

Disclaimer

The ThangTran 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
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 the following commit hash:

```
53ca9cb1
```

Scope

```
./src/
#-- PasswordStore.sol
```

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

Roles

- Owner: The user who can set and read the password.
- Outsides: No one should be able to set or read the password.

Executive Summary

Issues found

Severity	Number of issue found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password onchain 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` variable is intended to be a private variable and only access through `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 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's 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 output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And we get an output of:

```
myPassword
```

Recommended Mitigation: Due to the fact the every data is public on the blockchain. The smart contract architecture should be rethought. I suggest 2 ways of doing that: 1. You can store the password in a backend. 2. Password should be encrypt before push to blockchain. In this way, user must be remember another password to decrypt the password. Also, i suggest remove the view function so others person can't view the transaction on blockchain.

[H-2] Everyone can set/change the password of the contract, severly breaking the contract intended functionality

Description: The `PasswordStore::setPassword` function is set to be an `external` function and accesible for everyone, however, the 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 the password of the contract, severly breaking the contract intended functionality.

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

Code

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

```
if(msg.sender != owner){
    revert PasswordStore__NotOwner();
}
```

Medium

Low

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that dosen't exist

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) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec say it should be `getPassword(string)` **Impact:** The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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

Gas