PasswordStore Audit Report

Version 1.0

Traxler Security Services

December 26, 2023

PasswordStore Audit Report

Traxler Security Services

December 26, 2023

Prepared by: [Traxler Security Services] Lead Security Researchers:- - Traxler

Table of Contents

- Description:
- Impact:
- Proof of Concept:
- Recommended Mitigation:
    - Informational
- [I-1]
- Impact
- Recommended Mitigation

Protocol Summary

PasswordStore is a protocoldedicated to storage and retrieval of user's passwords. The protocol is designed to be used by single user and not designed to be used by multiple users. Only the owner is able to set and retrieve password.

Disclaimer

The Traxler Security Services 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 described in this document corresponds to the following commit hash

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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

The team reviewed this protocol for 1 hours using various tools like Solidity Metrics, Cloc etc to have an idea what does this protocol do and found some really serious vulnerability.

Issues found

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

Findings

High

[H-1] Storing the password on-chain 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
"PasswordStrore::s_password" variable is intended to be private
    variable and only be accessed by the "PasswordStore::getPassword()"
    function,
which is intended to be only called by the owner.

We show one such method of reading any data off-chain below.
```

Impact:

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

Proof of Concept:

```
1. Make 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 "PasswordStore::
        s_password" in the contract.
    ```
    cast storage <CONTRACT_ADDRESS> 1 --rpc-url http://127.0.0.1:8545
    ```

    You can parse that hex to string:
```

```
18      ```
19      cast parse-bytes32-string 0
           x6d7950617373776f726400000000000000000000000000000000000000000014

20      ```
21
22      And get an output:
23      ```
24      myPassword
25      ```
```

Recomended Mitigation:

```
1  Due to this, the entire architecture of the contract must be rethought.
       One could encrypt the password on-chain and
2  then store the encrypted data on-chain. This would require the user to
       remember another password offchain to decrypt
3  the password. However, you'd also likely to want to remove the view
       function as you wouldn't want the user to
4  accidentally send a transaction with the password that decrypts your
       password.
```

[H-2] `PasswordStore::setPassword()` has no access controls, meaning it can be accessed by a non-owner.

Description:

This function is set to be an 'external' function, however, the natspec and the overall purpose of this smart contract is that "IT ALLOWS ONLY THE OWNER CAN SET PASSWORD". This defined function has no access control syntax to restrict non-owners from utilizing this function.

```
1    function setPassword(string memory newPassword) external
2  @>    //@audit:- This function has no access controls defined.
3    {
4        s_password = newPassword;
5        emit SetNetPassword();
6    }
```

Impact:

```
Anyone can access the this function and set the password other than the
    owner.
```

Proof of Concept:

Adding the following to the “passwordstore.t.sol” :-

```
    function test_anyone_can_set_password(address randomAddress)
        external
    {
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword2 = passwordStore.getPassword();
        assertEq(actualPassword2, expectedPassword);
    }
```

Recommended Mitigation:

Add access control specifier to the function `passwordStore::setPassword()`.

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

Informational

[I-1]

The `PasswordStore::getPassword()` function is getPassword() while the natspec says it should be `getPassword(string)`

Impact

The natspec is incorrect.

Recommended Mitigation

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

‘‘‘