Protocol Audit Report

Version 1.0

Cyfrin.io

April 21, 2024

PasswordStore Audit Report

toshiiki

March 7, 2023

Prepared by: toshiiki Lead Security Researcher: - xxxxxxx

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, and is not designed to be used by multiople users. Only the owner should be able to set and access thsi password.

Disclaimer

The YOUR_NAME_HERE 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 to 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	# of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] password stored on-chain is visible to anyone and no longer private

Description: All data stored 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 intend4ed 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 thhe private password, severely breaking the functionality of the protocol.

Proof of Concept:(proof of code)

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

```
1. Create a locally running chain
2. Deploy the contract
3. Run the storage tool to identify storage slot 1 of the contract
4. you will get an ouutput that looks like this: 0
   x6d7950617373776f726400000000000000000000000000000014
5. parse the output hex value to a string and get the unencrypted
   password
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, then store the encrypted password on-chain. This would requirethe user to remember another password off-chain to decrypt the password. However, you would also likely want to remove the view function as you wouldnt want the user to accidentally send a transaction with the password that decrypts the encrypted password.

[H-2] `PasswordStore::setPassword` has no access controls, function can be accessed and changed by anyone

Description: The `PasswordStore::setPassword` function is intended to be only called by the owner of the contract, however, there is no require or assert statement to prevent other users from accessing the function and changing the password.

```
    function setPassword(string memory newPassword) external {
> > >   // @audit - missing access control - any user can set password
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password of the contract, severely 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();
```

```
9           assertEq(actualPassword, expectedPassword);
10      }
```

Recommended Mitigation: Add an accesss control conditional to the `setPassword` function

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist, making the natspec incorrect

````
**Description:**
```javascript
    //* @param newPassword The new password to set.
    function getPassword() external view returns (string memory) {
        return s_password;
    }

```
the `PasswordStore::getPassword` function signature is `getPassword()` which the natspec says should be `getPassword(string)`.

**Impact:** The natspec is incorrect.

**Recommended Mitigation:** remove the  incorrect natspec line
```diff
-     * @param newPassword The new password to set.
+
```
````