Protocol Audit Report

Version 1.0

Saikumar

February 19, 2024

Protocol Audit Report

Saikumar

February 12, 2024

Prepared by: Saikumar Lead Auditors: - Saikumar

Table of Contents

Protocol Summary

A smart contract applicatoin 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 Saikumar 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 are correspond to the following 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 about how thw audit went

Issues found

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

Findings

High

[H-1] Storing password on chain makes it visible to anyone and no longer private

Description: All data stored on chain is visible to anyone. In the contract the variable `PasswordStore::s_password` is intended to be a private variable which is only seen by the owner by using the `PasswordStore::getPassword` function to view the password stored.

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

Impact: Anyone can see the password stored severly 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.

```
make anvil
```

2.Deploy the contract using the make file

```
make deploy
```

3.Run the storage tool

We use slot 1 because in out contract `s_password` is stored in slot 1

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

You'll get a output which looks like this

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can parse the hex to string with below command

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

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

[H-2] `PassswordStore::setPassword` is missing access control, leading to non-owner set password

Description: The `PasswordStore::setPassword` is used to setPassword and is a `external` function, however as natspec of the function and the purpose of this function in the smart contract `This function allows only the owner to set a new password.`

```
1    function setPassword(string memory newPassword) external {
2  @>      //@audit - There are no access control
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: Anyone cans set and change the password which severely breaks the contract intended functionality

Proof of Concept: Add the following to `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 the access control conditional to `setPassword` function

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

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)
```

The `PasswordStore:getPassword` signature is `getPassword` but as said signature according to the natspec is `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect line

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