Protocol Audit Report

Version 1.0

Cyfrin.io

October 3, 2024

PasswordStore Audit Report

CC

October 3, 2023

Prepared by: CasinoCompiler Lead Auditors: - CC

Table of Contents

Protocol Summary

The protocol is designed to allow a singular user save a password and also retrieve it. The purpose was so that it was only visible to the that singular user who has full access control.

Disclaimer

The CC 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 correspond to the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
==> PasswordStore.sol
```

Roles

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

Executive Summary

As is, the current architecture of the protocol should not be implemented and be redesigned. By nature, all data on a blockchain is public information, thus, sensitive data should not be stored in raw form.

Issues found

Severity	Number of Issues Found
High	2
Medium	0
Low	0
Informational	1
Gas	0
Total	3

Findings

High

[H-1] Variables stored on-chain are not hidden; anyone can find the password.

Description:

One use-case of the smart contract `PasswordStore.sol` is to retrieve `PasswordStore.sol::s_password` and only the user is able to retrieve this password. This function (`PasswordStore.sol::getPassword()`) is implemented correctly, but the purpose of the function as stated by the documentation is to *hide* the password.

Solidity keywords of public, private, external, internal only specify an allowed contract-to-contract interaction.

Variables stored on chain are visible to everyone at all times.

Therefore, the password is obtainable by anyone the moment the contract is deployed.

Proof of concept is stated below.

Impact: Anyone can read the private password, severly breaking the functionality of the contract.

Proof of Concept:

1. Create a locally running chain.

```
make anvil
```

2. Deploy the contracts.

```
make deploy
```

3. Run the storage tool

```
cast storage <Passwordstore_contract_address>
```

4. Copy the hex value associated with s_password.
5. Use command to convert hex value to string

```
cast parse-bytes32-string <copied_hex_value>
```

6. Password is now visible.

Recommended Mitigation:

sensative data should not be stored on a blockchain. the use of a blockchain is to verify ownership, and your “password” is your private-key.

use an offline password storage system. This protocol / contract should not exist.

If you wanted to create some kind of 2-stage encryption password with the decryption key offline, that is a different protocol altogether; additionally, you would have to ensure the decryption key is robust as someone crazy would eventually be able to decrypt your password.

[H-2] `PasswordStore.sol::setPassword()` lacks correct access control, so anyone can set the password.

Description:

Another use case of the protocol is to set a password. Each time `PasswordStore.sol::setPassword()` is called, the password can be changed. The purpose is so that only the `PasswordStore.sol::s_owner` is able to set the password.

no checks in place for intended purpose.

Proof of concept is stated below.

Impact:

Anyone can change the password.

Proof of Concept:

Add the following test to `PasswordStore.t.sol`:

Code

```
    function test_AnyoneCanChangePassword(address randomAddress) public {
        // Pick arbitrary password
        string memory changedPassword = "changed password";

        // Simulate non-owner calling setPassword()
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        passwordStore.setPassword(changedPassword);
        vm.stopPrank();

        // Retrieve password by calling getPassword with owner
        vm.startPrank(owner);
        string memory actualPassword = passwordStore.getPassword();
        vm.stopPrank();

        // Assert password is equal to value alice set.
        assertEq(actualPassword, changedPassword);
    }
```

Recommended Mitigation:

Add access control check at the beginning of `PasswordStore.sol::setPassword()`:

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

Informational

[I-1] Documentation states a non-existent @param

Description:

```
1
2       /*
3        * @notice This allows only the owner to retrieve the password.
4    @>  * @param newPassword The new password to set.
5        */
6       function getPassword() external view returns (string memory) ...
```

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

Impact:

incorrect natspec

Recommended Mitigation:

Remove incorrect natspec line

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3    -   * @param newPassword The new password to set.
4        */
```