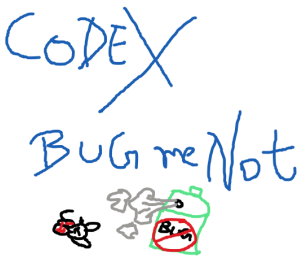

Protocol Audit Report

Version 1.0

Cyfrin.io

January 9, 2024

Protocol Audit Report

CodexBugMeNot

January 9, 2024

Prepared by: [CodexBugMeNot] Lead Auditors: - CodexBugMeNot

Table of Contents

Protocol Summary

Password Store is a smart contract application 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 CodexBugMeNot 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:

```
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 audit was thorough and went very well,the audit was conducted for 9 hours and we used standard tools to conduct the audit

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 private variable `PasswordStore::s_password` is intended to be private and only be accessed by the owner of the contract using `PasswordStore::getPassword` but it is not the case as it can be read by any one on-chain.

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

Impact: Any one can Read the private password,severely breaking the functionality of the protocol.

Proof of Concept:(Proof of Code) The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

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'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000`

You can then parse that hex to a string with:

```
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 password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no Access Control , which will allow a non-owner to change password

Description: The `PasswordStore::setPassword` function is set to be an `external` function but the natspec comments mention `This function allows only the owner to set a new password.` which allow anyone to set the password.

```
function setPassword(string memory newPassword) external {
  -----> // @audit There are no Access Controls
        s_password = newPassword;
        emit SetNetPassword();
```

```
5       }
```

Impact: Anyone can Change/Set the password using the `PasswordStore::setPassword` function ,severely breaking the Contract intended functionality.

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

Code POC

```
1   function test_anyone_can_set_password(address randomAddress) public {
2           vm.assume(randomAddress != owner);
3           vm.prank(randomAddress);
4           string memory expectedPassword = "myNewPassword";
5           passwordStore.setPassword(expectedPassword);
6
7           vm.prank(owner);
8           string memory actualPassword = passwordStore.getPassword();
9           assertEq(actualPassword, expectedPassword);
10      }
```

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

```
1  if(msg.sender!=owner){
2    revert PasswordStore_NotOwner();
3  }
```

Informational

[I-1] The `PasswordStore::getPassword` Natspec indicates a parameter `newPassword` which doesn't exist causing the natspec to be incorrect.

Description:

```
1   /*
2       * @notice This allows only the owner to retrieve the password.
3   ---> * @param newPassword The new password to set.
4       */
5       function getPassword() external view returns (string memory) {
6           if (msg.sender != s_owner) {
7               revert PasswordStore__NotOwner();
8           }
9           return s_password;
10      }
```

the `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says 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.
```