Protocol Audit Report

Version 1.0

Cyfrin.io

March 25, 2024

Protocol Audit Report

Frankline Mukopi

March 7, 2023

Prepared by: Frankline Lead Security researcher:

- Frankline Mukopi

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
- Medium
- Low
- Informational
- Gas

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 multiple users. Only the owner should be able to set and access this password.

Disclaimer

A security audit by the 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 the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc
./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

Add notes about how the audit went, types of things you found etc. We spent X hours with Z auditors using Y tools. etc

Issues found

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

Findings

High

[H-1] Variables stored in storage on-chain are visible to anyone, no matter the solidity visibility keyword meaning the password is not actually a private password.

Description: All data stored on-chain 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 intended 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 the password severly breaking the functionality of the contract.

Proof of Concept: (Proof of code) The below test shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to chain.

```
make depoy
```

3. Run storage tool We use 1 because that's the storage slot for `s_password` in the contract.

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

You'll geta n output that looks like this:

`0X6d790617373776f726400000000000000000000014`
```

You can then parse that hex to a string with:

```
cast parse-bytes32-string 0X6d7950617373776f726400000000014
```

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 would'nt want the user to accidentally send a transaction with the password that decrypts your password.

Likelihood and Impact: -Impact:HIGH -Likelihood: HIGH -Severity: HIGH

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password.

Description: the `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overall purpose of the smart contract is that `This function allows only the owner to set the password.`

```
  function setPassword(string memory newPassword) external {
        // @audit - There are no access controls.
```

```
3           s_password = newPassword;
4           emit SetNetPassword();
5       }
```

Impact: Anyone can change the password severly breaking the contact's intended functionality.

Proof of concept: Add the following into the `PasswordStore.t.sol` test file.

Code

```
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 `setPassword` function.

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

Likelihood and Impact: -Impact: HIGH -Likelihood: HIGH -Severity: HIGH

Informative

[I_1] The `PasswordStore::getPassword` natspec indicates a parameter that 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       // @audit there is no new password parameter.
6
7       function getPassword() external view returns (string memory)
```

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

Impact: The natspec is incorrect. **Proof of concept:** No proof of concept for this. **Recommended mitigation:** Remove the incorrect natspec line.

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

Likelihood and Impact: -Impact: NONE -Severity: Informational/Gas/Non-crits