Protocol Audit Report

Version 1.0

Kamau Audits

February 12, 2024

Protocol Audit Report

Kamau Audits

February 12, 2023

Prepared by: Kamau Audits Lead Security Researcher: - James Kamau M

Table of Contents

    * [I-1] The `PasswordStore::getPassword` natspec indicates a paramater that doesn't exist causing the natspec to be incorret

  - Likelihood and Impact:

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

The Kamau Audits 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 the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

- Owner: Is the only one who should be able to set and access the password. For this contract, only the owner should be able to interact with the contract.
- Outsiders: No-one else should be ble to set or read the password

Executive Summary

We spent 16 hours in the audit with James and Patrick working on the protocol using fuzz test tools. We found some issues which are documented below

Issues found

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

Findings

High

[H-1] On-chain storage of passwords makes it visible to anyone. This means the password is not private and retrievable by anyone

Description:
The data set in `PasswordStore::s_password` is stated as private but when the contract is deployed on chain the password ceases to become private as on-chain data can be read by anyone in theblockchain. This leds to the retrival pf the password to become easy for anyone during the function call of `PasswordStore::getPassword` which is only intended by the owner of the password

Impact:
Anyone can read and retrieve the private password and thus severely breaking the functionality of the protocol.

Proof of Concept: 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 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:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation:
Encryption can be done off-chain and then store the encrypted password on-chain and this would require the user to remember the password set for encryption. Also the view function might be better without it as the user might acciedentally send a transaction with the password that decyrpts the 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 function is set to be external and this means it can be called by anyone and to top it off there are no conditions placed to ensure that the function is only called by the owner. This means another user can change the password placed.

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

Impact:
Anyone can set/change the password of the contract and thus severely breaking the functionality of the contract

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

Code

```
    function test_non_owner_can_set_password(address  randomAddress)
        public {
        vm.assume(randomAddress != owner);
        vm.pank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

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

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

Likelihood and Impact:

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

Informational

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

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` function signature is `getPassword()` while natspec says it should be `getPassword(string)`

Impact:
Natspec is incorrect

Recommended Mitigation:
Remove the incorrect natspec line.

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

Likelihood and Impact:

-Impact : None -Likelihood : HIGH -Severity : Informational/Gas/Non-Crits