Protocol Audit Report

Version 2.0

D.E.C.N Group

February 26, 2024

Protocol Audit Report

k2ekimdam

Febrero 25 2024

Prepared by: k2ekimdam Lead Auditors:

-

Table of Contents

Protocol Summary

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 D.E.C.N. 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:

```
Commit Hash:
```

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

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 `PasswordStored::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 private password, 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 de 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:

Due to this,the overall architecture of the contract should be rethougth. 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 accidentally send a transaction with the password that decrypts your password.

Likelihood & Impact:

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

[S-#] `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 a new password`.

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

Impact: Anyone can set/change password of the contract, severly breaking the contract intended functionality.

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

```
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 != owner) {
2          revert PasswordStore__NotOwner();
3      }
```

Likelihood & Impact:

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicate 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       function getPassword() external view returns (string memory) {
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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

Likelihood & Impact:

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