Protocol Audit Report

Version 1.0

Herman Effendi

July 30, 2024

Protocol Audit Report

Herman Effendi

July 30, 2024

Prepared by: Herman effendi

Table of Contents

Introduction

The Kupu 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.

Prepared by: Herman effendi

Lead Auditor Security: - Herman effendi

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

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

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

Findings

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

High

[H-1] Storing password on-chain makes it visible to everyone and no longer private Description:

All data stored on-chain is visible for everyone, and can be read directly from 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 how to read the data off-chain below.

Impact:

Anyone can read the private password, severly breaking the functionality of the protocol

Proof of Concept:

1. Create locally running on-chain

```
make anvil
```

2. Deploy the contract on anvil local chain

```
make deploy
```

3. Get the address contract and get value from contract storage through command `cast`

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
    http://127.0.0.1:8545
```

4. Decode the value from, using command `cast`

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014

myPassword // output
```

5. finally we can read the actual password from local on-chain anvil.

Recommended Mitigation:

Due to this, the overall the architecture of the contract should be rethought. One could encrypt the password off-chain, and store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the password. however, you would also likely want to remove the view funtion as you wouldn't be want the user to accidently send a transaction with the password that decrypts the password.

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

The `Password::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 the password of the contract. severly breaking the contract intended functionality.

Proof of Concept:

add the following to the `PasswordStore.t.sol` test file.

Code

```
    function test_non_owner_can_setting_password(address random) public
        {
```

```
        vm.assume(random != owner);

        vm.prank(random);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();

        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation:

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

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter doesn't exist, causing the natspec to be incorrect. 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) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

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

Impact:

The natspec is incorrect.

Proof of Concept:

Remove the incorrect natspec line.

Recommended Mitigation:

Remove the incorrect natspec

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