Protocol Audit Report

Juan Pedro Ventura

October 25, 2024

Prepared by: Fishy Lead Auditors:

- Juan Pedro Ventura (Fishy)

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to store and retrieval of a user's passwords. The protocol is designned 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 Juan Pedro Ventura 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

		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 :

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
└── PasswordStore.sol
```

Roles

```
Owner: The user who can set the password and read the password.
Outsides: No one else should be able to set or read the password.
```

Executive Summary

I spend 2 hours on this audit, using the following tools:

- Foundry test suite
- Cast
- Anvil

Issues found

Severity	Number of issues found
High	2
Medium	0

Severity	Number of issues found
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visable to anyone, and no longer private

Description: All data stored on-chain is visable to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only acceessed through the `PasswordStore::getPassword` function. Wich is intended to be only called by te owner of the contract.

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

Impact: Anyone can read the private pasword, 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 the blockchain.

1. Start a anvil local environment

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

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

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. Decode the hex value

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

Recommended Mitigation: Due this issue, all the protocol arquitecture should be rethought. What i would do is to have a password, or unique identifier for each password, and to get the password, user should send to a funcion this password hash, and this funcion is going to return the password.

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

Description: The `PasswordStore::setPassword` Should be only callable by the owner of the contract, but this funcion, does no have any access control.

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

Impact: Any user could change the password, and this will break the protocol.

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

▶ Code

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

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

        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: This issue is making this protocol not secure, a validation, that validates if the msg.sender is the owner should be added to the function.

```
    // here is the validation that you should add
    if(msg.sender != _owner) {
        revert;
    }
```

Informational

[I-1] `PasswordStore::getPassword` natspec a parameter that doesen't exist, causing the natspec to be incorrect

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
    *  // @audit v there is no newPassword parameter
@>     * @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 `PaswordStore::getPassword` signature is `getPassword()` while the natspec says it should be `getPassword(string)`

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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