Protocol Audit Report

Version 1.0

Ease Industries

June 20, 2024

Protocol Audit Report

Ease Industries

June 20, 2024

Prepared by: Ease Industries

Lead Auditors: - Eyan Ingles

Table of Contents

Protocol Summary

The PasswordStore Protocol is designed to leverage the blockchain ledger for storing passwords on-chain. This protocol enables users to gain access to the stored information by invoking specific functions. By utilizing blockchain technology, PasswordStore ensures that password management is both reliable and transparent.

Disclaimer

The Ease Industries 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 to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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

We have dedicated approximately 2 hours to manually review the scope of the coding and an additional 20 minutes to automated testing using Foundry.

Detailed specifications of Foundry testing scopes are provided in the Findings section.

During our manual code review, we identified several significant issues within the protocol. For a comprehensive overview of the problems discovered, please refer to the #Findings section.

Issues found

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

Findings

High

[H-1] Storing any passwords on-chain makes it visible to anyone, exposing passwords.

Description: All data stored on-chain is visible to anyone and can be read directly from the blockchain, the `PasswordStore::s-password` 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, severaly breaking the functionality of the protocol.

Proof of Concept:

The test code below will show how anyone can read a private variable straight from the blockchain.

1. Create a localhost blockchain by using

```
anvil
```

2. using the following code:

```
cast storage <contractAddress> <storageSlot>
```

which should also look like

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

which then will give us an output of:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

this is now the storage information in `bytes`.

3. now we know the bytes, we now can convert it to a string gaining the stored varibale in string form by using:

```
cast parse-bytes32-string <bytesCode>
```

or

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

giving us an output of

```
myPassword
```

Recommended Mitigation: Due to the structure of the protocol has be build around, it is unreasonable for the protocol to try and store private information on-chain in this way, other ways that may be feasable would be ERC that allows for private use or exploring such areas as such.

[H-2] `PasswordStore::setPassword()` has no access control, meaning a non-owner can successfully use this function changing the password.

Description: The `PasswordStore::setPassword()` is set to `external` with no `if` or `require` statements to stop anyone accessing this function as it is stated that only owner can set newPassword.

```
    // error! doesnt have an address to assign the password too
    // error! does not have a only owner modifier
    // @audit any user can set a password
    // missing access control
    function setPassword(string memory newPassword) external {
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can call upon the `setPassword()` function, changing the password breaking the intended functionality of the contract.

Proof of Concept: Refer to code snippet below in the `Code` tab, by adding this code into the `passwordStore.t.sol` file and running the test:

```
forge test
```

Code

```
    function test_anyone_can_change_passeword(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: Add access control to prevent non-owners calling the `passwordStore::setPassword()` function.

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

Informational

[I-1] The `PasswordStore::getPassword()` natspec indicates a paramater that does not 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)
```

The function of `PasswordStore::getPassword()` function signiture is `getPassword()` instead the natspec suggests that it is `getPassword(string)`

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line

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