Protocol Audit Report

Prepared by: mddragon18

Table of Contents

Protocol Summary

The protocol , PaswordStore , is a smart contract that allows the users of the contract to store their password on chain in a safe and secure way , and also to retrieve their password whenever they want to, other users should not be able to retrieve the passwords of the users.

Disclaimer

The Dragon Security 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 in the document are based on the commit hash :

Commit Hash: 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

Scope

```
./src/
└── PasswordStore.sol
```

Roles

Owner - Can set and retreive the password
Outsiders - Cannot set or retreive the password set by the owner.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Informational	0

Findings

High

[H-1] Variables stored on chain are visible to anyone regardless of their visibility. Meaning the password is not private.

Description: All data on the blockchain can be read by anyone and the variable `PasswordStore::s_password` used to store the password can be read by anyone from the chain regardless of its private nature.

Impact: Anyone can read the password stored by an user , which break the whole functionality of your protocol.

Proof of Concept:

Follow the below process to replicate the vulnerability

1. Set up a local anvil chain

```
make anvil
```

2. Deploy the contract

```
make deploy
```

3. Enter the address of contract and slot of the memory (1 in this case).

```
cast storage <contract address> <slot> --rpc-url <url>
```

4. Input the hex from step 3.

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

5. You will find the password.

Recommended Mitigation: Due to this , the overall architecture of the contract should be rethought. You encrypt the password offchain and store it in the contract. This would require the user to remember another password that decrypts the password.

[H-2] `PasswordStore::setPassword()` has no access controls.

Description: The `PasswordStore::setPassword()` function has not been given any access control while being an `external` function, and anybody can set the password overridding the user set password. Which goes against the protocol. `This function allows only the owner to set a new password.`

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

Impact: Anyone can set password of the user , which causes the user to lose the password permamently.

Proof of Concept: Add the following to your test file.

▶ Code

```

```

```
function test_anyone_can_set_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 to the function.

```
function setPassword(string memory newPassword) external {
 >>     if(msg.sender != owner)
        revert PasswordStore__NotOwner();
        s_password = newPassword;
        emit SetNetPassword();
    }
```