PasswordStore
Protocol Audit Report

Version 1.0

Oleh Yatskiv at OmiSoft

May 8, 2024

PasswordStore Protocol Audit Report

Oleh Yatskiv at OmiSoft

May 8th, 2024

Prepared by: OmiSoft

Lead Auditor: Oleh Yatskiv

Table of Contents

Protocol Summary

PasswordStore is a protocol that is designed for the user to save and retreive their password. The user (in other words: the deployer) of the smart contract should be the only one to be able to use contract's functionality. Other users should have no access to the contract's data of a particular user.

Disclaimer

Oleh Yatskiv from the OmiSoft 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 auditor 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 to the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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

Storing the password on-chain makes it visible to anyone, and no longer private

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be priate variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract. However, the password is stored on-chain, and can be read by anyone.

We show one such method of reading any data off-chain in the Proof of Concept section below.

Impact: Anyone can read the password stored in the contract, severely breaking the functionality of the protocol.

Proof of Concept: The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain:

```
anvil
```

2. Deploy the contract:

```
make deploy
```

Get the contract address to use for the next step:
`0x5FbDB2315678afecb367f032d93F642f64180aa3`

3. Get the password from the contract by accessing the 1st storage slot (that's where `s_password` is stored) using the following command:

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

You should get the bytes representation of the password:
`0x6d7950617373776f72640000000000000000000000000000000000000000000014`

4. Parse the bytes to string to get the password:

```
cast parse-bytes32-string 0
    x6d7950617373776f72640000000000000000000000000000000000000000000014
```

And you get the password: `myPassword`

Recommended Mitigation: Taking everything in consideration, the password should not be stored on-chain, and the whole architecture of the contract should be rethought. One could encrypt the password off-chain and store the encrypted password on chain, but that would require the user to store another password off-chain. However, you would also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts user's password.

`PasswordStore::setPassword` has no access control leading to anybody being able to change the owner's password

Description: Due to missing access control on the `PasswordStore::setPassword` function, anyone can change the password of the owner of the contract. This is a severe security issue, as according to natspec for the function: `This function allows only the owner to set a new password`, which it doesn't.

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

We show how anyone could change the password in the Proof of Concept section below.

Impact: Anyone can set the password of the owner of the contract, breaking the core functionality of the protocol.

Proof of Concept: The below test code shows how anyone can set the password.

The following test added to `PasswordStore.t.sol` passes:

```
    function test_fuzz_anyone_can_set_password(address randomAddress)
       public {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);

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

        vm.startPrank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Which means that any account can set the password of the owner.

Recommended Mitigation: It is necessary to add some access control to the `PasswordStore::setPassword` function, that will make sure that only `PasswordStore::s_owner` can call the function. Here's one of the ways it could be done:

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

Informational

Natspec of the `PasswordStore::getPassword` function is misleading, which can confuse developers

Description: The natspec of the `PasswordStore::getPassword` function suggests that there should be a parameter called `newPassword`, when in reality the function takes no parameters, which is misleading.

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
        ...
    }
```

Impact: It can lead to confusion among the developers using the function

Recommended Mitigation: Remove the `* @param newPassword The new password to set.` line to make the natspec for the `PasswordStore::getPassword` function correct.