Protocol Review Report

Version 1.0

https://github.com/mighty-hotdog

June 24, 2024

Password Store Application Audit Report

@mighty_hotdog

June 24, 2024

Prepared by: @mighty_hotdog

Lead Security Researcher: @mighty_hotdog

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
  - Informational

Protocol Summary

This is a smart contract application for storing a password. Users are able to store a password and then retrieve it later. Other users should not be able to access the password.

Disclaimer

The security researcher 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 this report are applicable to the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

1. Owner: The user who can set the password and read the password.
2. Other Users: No one else should be able to set or read the password.

Executive Summary

A summary of what happened in the security review.

Issues found

Severity	Number of Issues Found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password in-the-clear in an onchain variable `PasswordStore::s_password` makes it visible to anyone Description: The (unencrypted, unhashed) password stored in the `PasswordStore::s_password` variable is intended to be accessible only by the owner. But since it is a variable stored onchain, it is actually visible to anyone.

Impact: The password stored by the owner is not private. Anyone can see it. This severely breaks the intended functionality of this contract.

Proof of Concept: Step #1 - Start up Anvil

```
make anvil
```

Step #2 - Deploy PasswordStore contract onto Anvil

```
make deploy
```

Step #3 - Use `cast storage` and `cast parse-bytes32-string` to view the password

```
cast storage <address of deployed contract> 1 --rpc-url http://
    127.0.0.1:8545
```

Note: - 1 is the storage slot of `PasswordStore::s_password` - `http://127.0.0.1:8545` is the rpc-url of the Anvil local chain -

may be obtained from the Anvil console printout upon deployment

Output looks like this:

```
0x123e423wecmfwq43c4fmwfj
```

Use `cast parse-bytes32-string` to view the password in human-readable form.

```
cast parse-bytes32-string 0x123e423wecmfwq43c4fmwfj
```

Output obtained:

```
myPassword
```

Recommended Mitigation: This protocol design is unsuitable for storing private passwords on-chain such as to be inaccessible to anyone other than the owner. A possible alternative is for the `PasswordStore::setPassword` function to encrypt the password with the owner's private key before storing the encrypted hash onchain. The owner may then call the `PasswordStore::getPassword` function with her private key to retrieve the password in its unencrypted form.

[H-2] Missing access controls: the `PasswordStore::setPassword` function allows anyone, not just the owner, to be able to set a new password

Description: The `PasswordStore::setPassword` function lacks the necessary access control to restrict caller access to only the owner. This makes it possible for anyone to call this function and set the password.

Impact: Anyone can set a password, or change the password set by the owner. This severely breaks the intended functionality of this contract.

Proof of Concept:

Add the following test to `PasswordStore.t.sol` and run it. This test will pass.

```
    function test_nonowner_can_set_password(address randomUser) public {
        vm.assume(randomUser != owner)
        string memory expectedPassword = "myNewPassword";
        vm.prank(randomUser);
        passwordStore.setPassword(expectedPassword);
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation:

Add the necessary access controls to the `PasswordStore::setPassword` function to restrict access to only the owner.

```
    function setPassword(string memory newPassword) external {
        //
           ////////////////////////////////////////////////////////////////

        // add this line for access control
        //
           ////////////////////////////////////////////////////////////////

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

        s_password = newPassword;
        emit SetNetPassword();
    }
```

Informational

[I-1] Natspec-to-code mismatch: the `PasswordStore::getPassword` function does not accept a `newPassword` input parameter as the function natspec says it does

Description: As per title.

Impact: Informational.

Proof of Concept: NA

Recommended Mitigation: The code correctly implements the intended functionality of the function. Correct the natspec to align it to the code.