PasswordStore Audit Report

Version 1.0

Jeremy Bru

February 17, 2024

PasswordStore Audit Report

Jeremy Bru

Feb 16, 2024

Minimal Audit Report - PasswordStore

Prepared by: Jeremy Bru (Link) Lead Security Researcher:

- Jeremy Bru

Contact: –

Table of Contents

Protocol Summary

- store any passwords that users register.
- Users should be able to store a password and then retrieve it later.
- Other persons than the user who registered a password, should not be able to access the password.

Disclaimer

I, Jeremy Bru, did 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

Uses the CodeHawks (Link) severity matrix to determine severity. See the documentation for more details.

Audit Details

Commit Hash: 7d55682ddc4301a7b13ae9413095feffd9924566

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

Issues found

Severyity	Number of findings
High	3
Medium	0
Low	0
Infos	2
———	———————
Total	4

Findings

High

[S-High1 [Critical]] No matter the visibility keyword, `PasswordStore:: s_password` is public and visible by anyone.

Description:

`PasswordStore:: private` keyword concerns readibility of the variable in the contract, but it is still visible by anyone. Which breaks completly the purpose of the protocol.

- `PasswordStore:: string private s_password;`

https://docs.soliditylang.org/en/v0.8.24/cheatsheet.html#function-visibility-specifiers

Impact:

No password is safe in the contract.

Proof of Concept:

Proof of Code showing how to read a private variable in a contract:

run a local chain

```
anvil
```

deploy contract

```
make deploy
```

Get storage value of `s_password`

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

returned bytes:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

Bytes to string

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

Returned string:

```
myPassword
```

Recommended Mitigation:

Other questions:

// q why us the password stored in a single variable ?

// q where is the mapping address to string for storing more than one password ?

// q why password is not hashed to be encrypted ?

Idea to mitigate the issue:

Architecture should be reworked. Password should be hashed and encrypted. A mapping should be used to store multiple passwords from many users as possible or for many password per users as possible.

Could encrypt off-chain and store the hash on-chain.

- Requires to remember another password.
- And remove the view function that should show the decrypted password.

As it is a whole rework of the architecture, I am not writing a fix for this issue.

[S-High2] Access control attack. Anyone can set a new password.

Description:

// q why anybody can set a new password ?

// @audit access control attack. high.

```
1      /*
2       * @notice This function allows only the owner to set a new
           password.
3       * @param newPassword The new password to set.
4       */
5      function setPassword(string memory newPassword) external {
6          s_password = newPassword;
7          emit SetNetPassword();
8      }
```

Impact:

Previously chosen password will be overwritten, and can be change by anyone.

Leading to loosing control over the password in one instant and deal with the consequences.

Proof of Concept:

Test Case `PasswordStore.t.sol`:

- When deploying the contract the password is set to `myPassword`.
- Using another address than the Owner address,I call the `PasswordStore::setPassword` function using a different password as input: `myNewPassword`.
- Then after, using the Owner address, I call the `PasswordStore::getPassword` function to retrieve the password.
- If the password is `myNewPassword`, then it is a success ↓(succeeded)

Code

```
    function test_anyone_can_change_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();
        console.log(actualPassword);
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation:

Could use a library like OpenZeppelin's Ownable to manage access control.

Could use a library like OpenZeppelin's Pausable to pause the contract in case of emergency.

Add a modifier or Only Owner check.

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

```
    /*
     * @notice This function allows only the owner to set a new
         password.
     * @param newPassword The new password to set.
     */
    function setPassword(string memory newPassword) external {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        s_password = newPassword;
        emit SetNetPassword();
    }
```

[S-High3] Ownership / Access control attack. Only owner can see password, users can't.

Description:

The Code documentations says:

`PasswordStore:: @notice This allows only the owner to retrieve the password.`

The project documentation says:

`Users should be able to store a password and then retrieve it later. Others should not be able to access the password.`

- So the owner should'nt be allowed to see users passwords.

Impact:

Password are not accessible by other users.

Leading to a possible password leaks by the owner.

```
1     function getPassword() external view returns (string memory) {
2         if (msg.sender != s_owner) {
3             revert PasswordStore__NotOwner();
4         }
5         return s_password;
6      }
```

Recommended Mitigation:

Rework the architecture or explicitly change the documentation.

Not safe for users.

Medium

- None

Low

- None

Informational

[S-Info1] Pragma version is outdated. Verify there is no breaking changes since then.

Description:

pragma version is not the most updated stable version:

// q is this the correct version ?

Impact:

Outdated version can lead to exploits and vulnerabilities that has been fixed in the latest version.

Contract: `PasswordStore.sol`

Proof of Concept:

0.8.18 -> 0.8.24

Recommended Mitigation:

0.8.18 -> 0.8.24

[S-Info2] Missing parameter or wrong description in getPassword function, add it or remove it.

Description:

// @audit there is no password to be set as parameter. q the below is needed ?

- `PasswordStore:: @param newPassword The new password to set.`

Impact:

Low impact if the description is wrong.

Proof of Concept:

remove this line -> `PasswordStore:: * @param newPassword The new password to set.`

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     */

function getPassword() external view returns (string memory) {
if (msg.sender != s_owner) {
// q only owner can read password ?
revert PasswordStore\_\_NotOwner();
}
return s_password;
}
```

Recommended Mitigation:

Add the parameter `PasswordStore:: newPassword` or remove it from the description.

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

Gas

- None