Logo

Protocol Audit Report

Prepared by: Mauro Júnior

Table of Contents

Protocol Summary

PasswordStore is a smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The Mauro Júnior 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 (https://docs.codehawks.com/hawks-auditors/how-to-evaluate-a-finding-severity) severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in the document correspond the following commit hash

Commit Hash

```

```

Scope

```
./src/
└── PasswordStore.sol
```

Roles

-Owner: The user who can set the password and read it -Outsiders: No one other than the user can set a new password

Executive Summary

The audit was not very long, spent 3 hours on it, found bugs and vulnerabilities that will be described in Issues Found. Used only Manual Review

Issues found

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

Findings

Highs

[H-1] Storing password on-chain are visible for anyone, meaning the password is not actually private

Likelihood & Impact

-Impact - HIGH -Likelihood- HIGH -Severity- HIGH

Description: All data stored on chain is visible from anyone and can be read directly from the blockchain, The 'PasswordStore:: s_password' variable is intended to be a private variable and only accessed through the 'PasswordStore:: getPassword' function, which is intended to only be 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, breaking the functionally of the protocol entirely.

Proof of Concept:(Proof of code) The below test shows that anyone can read the password directly from the blockchain

1. Create an locally anvil chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool Using 1 as the storage slot of `s_password` in the contract:

```
cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

After getting an output similar to this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can convert this hex into a string with

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

And you get the output:

```
Mypassword
```

Recommended Mitigation: Due to this problem, the overall architeture of the contract should be rethought. One could encrpyt the password off-chain, and store the encrypted password on-chain. This would require an user to remember the another password off-chain to decrpyt the password. Also removing the view function so that the user can't in accident send a transaction with the password that decrypts your password.

[H-2] `PasswordStore:: setPassword` has no access control, meaning non-owners could change the password

Likelihood & Impact

-Impact - HIGH -Likelihood- HIGH -Severity- HIGH

Description: The `PasswordStore:: setPassword` is set to be an external function, however, in the naspec's it says the overall purpose of the smart contract is that `This function should allow only the owner(set in msg.sender in constructor) to set an new password`

```
     function setPassword(string memory newPassword) external {
@>        // @audit there isn't access control here
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Impact: Anyone can change or set the password, severly breaking the contract intendeed functionality.

Proof of Concept: Add the following test in the `PasswordStoreTest.t.sol`

▶ Code

```
function testAnybodyCanSetPassword(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 conditional to the `setPassword` function.

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

Informational

[I-1] The `PasswordStore:getPassword` natspec indicates a parameter that is not in the code causing the natspec to be incorrect

Likelihood & Impact

-Impact - None -Likelihood- HIGH -Severity- Information/Gas/Non-Crit

Description

```
/*
     * @notice This allows only the owner to retrieve the password.
     //@audit there is no newpassword parameter!
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

The `PasswordStore:getPassword` function signature is `getPassword()` and it doesn't take newPassword as parameter so it should be `getPassword(string).`

Impact The natspec is incorrect

Recomended Mitigations Remove the incorrect natspec line or put the right parameters in the `getPassword()` function.

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