Password Store Audit Report

Version 1.0

Nacho Díaz

December 13, 2023

Password Store Audit Report

Nacho Díaz

December 13, 2023

Prepared by: Nacho Díaz Lead Auditors: - Nacho Díaz

Table of Contents

        - Description:
        - Impact:
        - Proof of Concept:
        - Recommended Mitigation: Add an access control conditional to the `setPassword` function
  - Informational
    * [I-1] The `PasswordStore::getPassword` natspect indicates there is a parameter `newPassword` that doesn-t exist therefore natspect is incorrect
        - Description:
        - Impact:
        - Recommended Mitigation:

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Nacho Díaz 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

A smart contract applicatoin 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.

Commit Hash : 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

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

How the audit went We spent X hours with Y auditors using Z tools.

Issues found

Severity	Number Of Issues
High	2
Medium	0
Low	0
Infornmational	1
Total	3

Findings

High

[H-1] TITLE: Storging plain text on-chain is vivible to anyone, then not private

Description: All data stored on-chain is visible to anyoune, and can be read by anyone. The `PasswordStore::s_password` in intended to be private and only accessed throug the `PasswordStore::getPassword` function. It is to other contracts but not to anyone who can read the blockchain.

Method to read it:

Impact: Cirtical. This bug breaks the functionality of the contract

Proof of Concept The below test cae shows how `PasswordStore::s_password` can be read by anyone.

1. Create a locally running chain

```
run anvil
```

2. In a new terminal deploy the contract and read the password

```
make deploy
cast storage "address of the deployed contract" "slot of the variable"
    --rpc-url http://127.0.0.1:8545
```

We receive a string like this: `0x6d7950617373776f726400000000000000000000000000000000000000000000` which is the hex representation of `myPassword`. We can convert it to a string using the following command:

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

Then we receive the password in plain text:`myPassword`

Recommended Mitigation: The best solution is to encrypt the password onchain and the store it inchain or directly store the password offchain cause the nature of the blockchain is to be public and transparent.

[H-2]`PasswordStore::setPassword` has not access control, a non-owner could change the password and set a new password

Description: Everybody can call the `PasswordStore::setPassword` function and change the password. The function does not have any access control. The `PasswordStore::setPassword` function is intended to be called only by the owner of the contract.

```
    function setPassword(string memory newPassword) external {
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Critical, this bug breaks all the funcionality of the contract allowing anyone to change the password

Proof of Concept: The below test case shows how `PasswordStore::setPassword` can be called by anyone:

Code

```
function test_not_owner_can_set_password() public {
        vm.startPrank(notOwner);
        string memory passowrd = "Im not the owner";
        passwordStore.setPassword(passowrd);
        vm.stopPrank();
        vm.startPrank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, passowrd);
    }
```

Recommended Mitigation: Add an access control conditional to the `setPassword` function

```
if(msg.sender!=owner) return;
```

Informational

[I-1] The `PasswordStore::getPassword` natspect indicates there is a parameter `newPassword` that doesn-t exist therefore natspect is incorrect

Description:

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

```
4       */
5      function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspect say it should be `getPassword(newPassword)`.

Impact: Incorrect Natspect

Recommended Mitigation: Remove the natspect line

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