Protocol Audit Report

Olaoye Salem(Nyxaris)

February 9, 2024

Prepared by: Olaoye Salem(Nyxaris)

Table of Contents

Protocol Summary

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.

Disclaimer

Olaoye Salem(Nyxaris) 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

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:

`2e8f81e263b3a9d18fab4fb5c46805ffc10a9990`

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 passowrd.

Issues found

Severity	Number of Issues Found
High	2
Medium	0
Low	0

Severity	Number of Issues Found
Info	1
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visible tp anyone and no longer private.

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

Likelihood and Impact:

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

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

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool We use `1` because that's the storage slot of `s_password` in the contract.

```
cast storage <ADDRESS_HERE> 1 --rpc url http://127.0.0.1:854
```

You'll get an output that looks like this

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with

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

ANd you get

```
myPassword
```

Recommended Mitigation:
Due to this, the overall architecture of the blockchgain should be rethought. One

could encrypt the password offchain and then store the encrypted password on-chain. This will require the user to remember another password offchain to decrypt the password. However, you would also likely want to remove the view function as you wouldn't wantthe user to accidentally send a transaction with password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no access controls. A non-owner can change the password.

Description: Since there is no check for the owner of the contract in the `PasswordStore::setPassword` function, a non-owner can change the passsword. Which is intended to be called by only the owner of the contract.

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

Impact: Anyone can set/change the password of the contract severely breaking the contract intended functionality.

Proof of Concept: Add the following to the `PasswordStore.t.sol` test file.

Code

```
function test_anyone_can_set_password(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(expectedPassword,actualPassword);
    }
```

Recommended Mitigation:
Add an access control to the `PasswordStore::setPassword` funtion.

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

Informational

[I-1] `PasswordStore::getPassword` indicates a parameter that doesn't exist, causing the natspec to be incorrect.

Description

```
/*
  * @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) {
         revert PasswordStore__NotOwner();
     }
     return s_password;
 }
```

The `Password::getPassword` function signature is `getPassword` while the natspec says it should be `getPassword(string)`

Recommended Mitigation Remove natspec line

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