PasswordStore Audit Report

Prepared by: Lhoussain Ait Aissa Lead Security Researcher:

Table of Contents

About LHOUSSAINE AIT AISSA

I am a dedicated Smart Contract Developer and Security Researcher with 2 years of hands-on experience in writing efficient and secure Solidity code. I possess strong expertise in auditing smart contracts to ensure robust security and functionality. I am seeking to leverage my technical skills and attention to detail to contribute to innovative blockchain projects within a dynamic team.

Disclaimer

LHOUSSAINE AIT AISSA makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the 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

Audit Details

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

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 password.

Executive Summary

Issues found

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

Findings

High

[H-1] Passwords stored on-chain are visible to anyone , no longer private

Description: All data on-chain are visible to anyone , so anuy use can read directly the data stored in the blockchain . the `PasswordStore::s_password` shoud read only on the owner using the `PasswordStore::getPassword` function , but in this case any one can get the password using the `Foundry cast` tool.

Impact: Password is visible to very body !!

Proof of Concept:

here is a few steps to improve how anyone can read this password in the storage using `Foundry cast` tool :

1. Create a locally runnig chain :

```
$ anvil
```

2. deploy `PasswordStore` contract in the local chain :

```
$ forge script script/DeployPasswordStore.s.sol:DeployPasswordStore --rpc-
url  http://127.0.0.1:8545 --private-key
0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80 --
broadcast
```

3. we use `1` because that's the slot of `s_password` in the contract

```
 $ cast storage <CONRTACT ADDR> 1 --rpc-url http://127.0.0.1:8545
```

You'll get output lik this :

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. convert the hex data to string

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

So the Output is the password :

`myPassword`

Recommended Mitigation: you shoud to encrypte the password to make it more deficulty to read in on-chain.

[H-2] Missing access control in `PasswordStore::setPasswrod` function, anyone could change the password

Description: Missing access control is high issus , because ant user can set/change the password . `PasswordStore::setPasswrod` function shoud revert any call form non-ower .

```
    function setPassword(string memory newPassword) external {
@>   // @audit - missing access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: anyone can set/change the password of the contract .

Proof of Concept:

Add the follow test to the `PasswordStore.t.sol` file test :

▶ Code

```
    function test_anyone_can_set_password(address randomAdd) public {
        vm.assume(randomAdd != owner);
        vm.startPrank(randomAdd);

        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

        vm.startPrank(owner);
```

```
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: to fix this issue , just add a access control condition shoud revert if is not the owner

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

Informational

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

Description: The natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

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

    function getPassword() external view returns (string memory) {
```

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line :

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