PasswordStore Audit Report

Prepared by: Julián Cabrera Marceglia

PasswordStore Audit Report

Prepared by: Julián Cabrera Marceglia

Lead Auditors:

- Julián Cabrera Marceglia (https://github.com/juliancabmar)

Assisting Auditors:

- None

Table of contents

▶ Detalles
See table

About Julián Cabrera Marceglia

I am a security researcher who want to make the web3 enviroment safer.

Disclaimer

The Julián Cabrera Marceglia team 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: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

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

Findings

High Risk Findings

[H-1] All data on-chain is public data, so everyone can see `s_password` value.

Description:
The `PasswordStore::s_password` variable is intented to be private and can only accessed by `PasswordStore::getPassword` function, but (how it show below) the value can be read directly from the blockchain data.

Impact:
Anyone can read the private password, severaly breaking the protocol's functionality.

Proof of Concept:

1. Running a local chain

```
$ anvil
```

2. Compile and Deploy the contract

```
$ make deploy
```

3. Get the storage slot #1 data (`s_password` data is on slot #1) for the deployed contract

```
$ cast storage [CONTRACT ADDRESS] 1
```

You'll get a result like this:
`0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. Parse the hex-bytes32 result to string

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

Getting the password value:
`myPassword`

Recommended Mitigation:
The actual contract objetive can't be reached, because you can't store a plain text password on-chain without that can be reading by everyone. A viable solution could be encript the password off-chain before store it on-chain, but this implies an extra step by the user.

[H-2] `PasswordStore::setPassword` haven't access control, meaning a non-owner could set a password.

Description:
The `PasswordStore:setPassword` function does not have any access verification mechanism that ensures only the owner can call it, this implies that everyone could set a password.

```
function setPassword(string memory newPassword) external {
@>  // @audit - There are not access controls
    s_password = newPassword;
    emit SetNetPassword();
}
```

Impact:
Anyone can set/change the current password of the contract, breaking severaly the intended contract functionality.

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

▶ PoC test:

```
function test_anyone_can_set_a_password(address anyone) public {
    string memory pass = "any password";
    vm.prank(anyone);
    passwordStore.setPassword(pass);

    vm.prank(address(owner));
    string memory actualPassword = passwordStore.getPassword();

    assertEq(pass, actualPassword);
}
```

Recommended Mitigation:
Add an access control condition on `PasswordStore::setPassword` function.

```
function setPassword(string memory newPassword) external {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    s_password = newPassword;
    emit SetNetPassword();
}
```

Informational Findings

[I-1] `PasswordStore::setPassword` function not take any parameters, but the related natspect indicated one.

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 `PasswordStore::setPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

Impact:
The natspec is incorrect.

Recommended Mitigation:

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