Protocol Audit Report

Prepared by: Sahil Kaushik

Table of Contents

Protocol Summary

This contract allows you to store a private password that others won't be able to see.

Disclaimer

I made 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

Likelihood Impact	High	Medium	Low
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 the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

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	1
Info	1
Gas Optimizations	0
Total	0

Findings

High

[H-1] Passwords are stored on chain are visible to anyyone irrespective of the visibility defined

Description: All data stored on chain are visble to anyone and can be readd direclty from the blockchain . The `PasswordStore::s_password` variable is a private variale and should be access by the `PasssowrdStore::getPassowrd` method only by their respective owners. But as of now it can be accessed by off chaian methodologies.

Impact: The password is not private

Proof of COncept: . 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://127.0.0.1:8545
```

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

Likelihood: High

Impact: High

Severity:High

[H-2] `PasswordStore::setPassowrd` has no access control , anyone can call it

Description `PasswordStore::setPassowrd` functin is set to be an external function and has no access contrrol and therefore anyone can call the function and set the password or change the passoword.

```
@>   // @audit-setPassword can be called by anyone
    function setPassword(string memory newPassword) external {
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Any one can call and set/change the password

Proof of concept: add the follwing in `Password.t.sol` test suit.

```
    function test_anyone_can_set_password(address anyone) public {
        vm.assume(anyone != owner);
        string memory newPassword = "hackedPassword";
        vm.prank(anyone);
        passwordStore.setPassword(newPassword);
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, newPassword);
    }
```

Recomended Mitigation Add an access control modifier to the function.

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

Likelihood: High

Impact: High

Severity:High

Medium

Low

Informational

[I-1] The `PasswordStore::getPassword` natspec 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) {
```

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()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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

Likelihood: N/A

Impact: N/A

Severity:High

Gas