title	author	date
PasswordStore Audit Report	Franklyn Ezeugonna	Febuary 1, 2024

PasswordStore Audit Report

Prepared by: Franklyn Ezeugonna Lead Auditors:

- Franklyn Ezeugonna

Assisting Auditors:

- None

Table of contents

About me

I'm passionate about uncovering vulnerabilities in systems and smart contract , always curious and eager to learn . Most importantly I love making new friends . feel free to reach out.

Disclaimer

I Franklyn Ezeugonna 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	1
Info	1
Gas Optimizations	0
Total	0

Findings

High

[H-1]Passwords stored on-chain are visable to anyone, not matter solidity variable visibility.

Description: All data stored on-chain is visible to anyone , and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended 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.

We show one such method of reading any data off chain below.

Impact: Anyone can read the private password, severely breaking the functionality of the protocol.

Proof of Concept:(Proof of Code)

The below test case shows how 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> I --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
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

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 remeber another password off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldnt want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` is callable by anyone

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however the natspec of the function and overall purpose of the smart contract is that `the function allows only the owner to set a new password.`

```
    function setPassword(string memory newPassword) external {
@>      // @audit:: There are no access controls
        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_any_one_can_set_password(address randomAdress) public {
```

```
        vm.assume(randomAdress != owner);
        vm.prank(randomAdress);
        string memory expected_password = "myNewPassword";
        passwordStore.setPassword(expected_password);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expected_password);
    }
```

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

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

Likelihood & Impact:

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

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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