Prepared by: [Alberto Guirado] Lead Auditors: Security

- xxxxxxx

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The YOUR_NAME_HERE team 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
	High	H	H/M	M
Likelihood	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 de described in this document correspond the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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 password

Executive Summary

Add some notes

*We spent X hours with Z auditors using Y tools

Issues found

Severtity	Numb of issues found
High	2
Medium	0
Low	1
Total	4

Findings

High

[H-1] Storing the password on-chain makes it visable to anyone, and no longer private.

Description: All data storage on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a rpivate variable and only access throgght the `PasswordStore::getPassword()` function, which is intended to be only called by the owner of the contract.

We show one such method of reding any data off chain below

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

Proof of Concept: (Proof of code)

```
The below test can shows how anyone can read the pass directly form 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 thats the storage slot of `s_password` in the contract

```
cast storage [] 1 --raddressContractpc-url [url]
```

You'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000`

Yoy can then parse that hex

```
cast parse-bytes32-string [hex_code]
```

Output

```
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 acccidentally send a transaction with the password that decrypt the password

Likelihood & Impact:

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

[H-2] `PasswordStore::setPassword` has no acces controls, meaning a non-owner could change the password

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

```
    function setPassword(string memory newPassword) external {
        // @audit - Ther no access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password of the contract, severly breaking the intended functionality

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

Code

```
function test_anyone_can_set_pass(address randomAddress)public {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory newp  ="hola";
        passwordStore.setPassword(newp);

        vm.startPrank(owner);
        string memory actualP  = passwordStore.getPassword();
        assertEq(actualP, newp);

    }
```

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

```
if(msg.sender != owner)_
    revert PasswordStore_NotOwner();
```

Medium

Low

Informational

[I-1] `Password::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.
     * /
```

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natpec line

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

Likelihood & Impact:

- Impact: NONE
- Likelihood: HIGH
- Severity: Informational/Gas/Non-crits

Informational: Hey, this isn't a bug, but you should know…

Gas