Password-Store Audit Report

Version 1.0

Ramprasad

June 27, 2024

Password-Store Audit Report

Ramprasad

june 27, 2024

Prepared by: Ramprasad

Table of Contents

    * [I-1] `PasswordStroe::getPassword` natspec indicats a parameter that doesnt exist, causing the natspec to be incorrect.
  - likelyhood and impact

Protocol Summary

Password store is a protocall dedicated to storage and retreval of a users passwords. the protocall is designed to be used by a single user, and is not desined to be used multiple users. Only the owner is able to set and access the password.

Disclaimer

The Ramprasad 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 described in this document corresponded the following commmit hash:**

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

-owner: The user who can set the password and read the password. -outsider: No one else should be able to set or read the password

Executive Summary

Issues found

Severtity	Number of issues found
High	2
Medium	0
Low	0
Information	1
Total	3

Findings

High

[H-1] Storing the password on onchain males is visible to anyone,and no longer private.

Description: All data stored on onchain is visble to anyone. and can read it directly from the blockchain. the `Passwordstore::s_password` variable is intended to be a aprivate variable and can access only through `Passwordstore::getPassword` function , which is intented to be only called by the owner of the contract.

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

Impact: Any one can read the private password directly from the blockchain.

Proof of Concept: (proof of code)

The below cast will show that any one can read the password directly from the blokchain.

1. Create a locally running chain.

```
make anvil
```

2. Deploy the contract on the chain. ‘`make deploy`
3. Run the storage tool.

we use `1` because thats the storage slot of the `s_password` in the contract.

```
cast storage <Address_here> 1 --rpc-url http://127.0.0.1:8545
```

you will get an out put looks like this: `0x6d7950617373776f72640000000000000000000000000000000000000000`

You can praise that hex to a string with:

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

And get an out put of:

```
myPassword
```

Recommended Mitigation: Due to this , the overall architechture of the cotract should be rethought. one could encrypt the password of-chain and then store the encrypted password on on-chain. This is would require the user to remember the another password of-chain to decrypt the password. However you would also likely want to remove the view function as you woudnt want the user to accidentally send a tracsaction with the password that decrypt your password.

likelihood & impact

-impact: High -likelihood: High -Severity: High

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

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 `This function allows only owner to set a password.`

"'java script function setPassword(string memory newPassword) external { @> // @audit There are no access controls. s_password = newPassword; emit SetNetPassword();

```

**Impact:** Any one can set/change the password of the contract.severly
    breaking the contract intended functionality.

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

<details>
<summary> CODE </summary>
``java script

function test_anyone_can_set_password(address randomAddress) public {
vm.assume(randomAddress != owner);
vm.prank(randomAddress);
string memory expectedPassword = "MynewPassword";
passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPssword = passwordStore.getPassword();
        assertEq(actualPssword, expectedPassword);
    }

``

</details>

**Recommended Mitigation:** Add an access controll condition to `
   setPassword` function:

```

~~~
```java script
if(msg.sender != s_owner) {
    revert PasswordStore_NotOwner()
}
~~~

likelihood and impact

-impact: High -likelihood: High -Severity: High

Informational

[I-1] `PasswordStroe::getPassword` natspec indicats a parameter that doesnt exist, causing the natspec to be incorrect.

Description:

`java script //@audit there is no newPassword parameter; * @param newPassword The new password to set. */ function getPassword() external view returns (string memory){`

the `PasswordStroe::getPassword` function signature is `getPassword()` which the natspec say it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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

likelyhood and impact

-impact: High -likelihood: Low -severity: Informational/gas