PasswordStore Audit Report

Version 1.0

Nathaniel

June 9, 2024

PasswordStoreAudit Report

Nathaniel Yeboah

June 9, 2024

Prepared by: Nathaniel Yeboah Auditor: - Nathaniel Yeboah

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    * [H-1] Storing a password on the blockchain make it visible to anyone, the password is not longer private
    * [H-2] `PasswordStore::setPassword` has no access control,which can make non-owner to set the password of the contract
  - Informational
    * [I-1] The nat spec document on `PasswordStore:getPassword` function indicates a parameter that does not exist, causing the natspec to be incorrect

Protocol Summary

This is a password store contract where users can store their passwords. Only the owner can set the password and view the password . The contract emits an event when a password is set.

Disclaimer

The Security Researcher 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 report are based on the code at the following commit hash: **

```
Commit hash: 7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

** The audit report is based on the following files: **

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

Roles

- **Owner:** The owner of the contract can set the password of the contract and view the password of the contract.
- **Others:** No one else can set the password of the contract or view the password of the contract.

Executive Summary

- Add some notes about how the audit went, what was found, types of things you found *
- We spend about 5 hours with 1 auditors using foundry *

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing a password on the blockchain make it visible to anyone, the password is not longer private

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

can only accessed throgh the `PasswordStore::getPassword` function , which is intended to be called by the owner of the contract only .

We show one such method of reading 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 case shows how anyone can read the password directly from the blockchain

1. Create a locall running chain

```
make anvil
```

1. Deploy the contract `PasswordStore`.

```
make deploy
```

1. Run the storage tool, to read the storage slot of the contract. We use 1 because the `s_password` variable is stored at that storage slot

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

You wil get an output that looks lie this: `0x6d7950617373776f726400000000000000000000000000000000`

You can then parse that hex to a string with this command:

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

You will get an output of `myPassword`

Recommended Mitigation: Due to this , the overall archieture of the contract should be rethought. One can encrypt the password before storing it on the blockchain, and then decrypt it when needed.This would require the user to remember another password off chain to decrypt the onchain password.How you also want ot remove the view function as you wouldnt wantthe user to accidentally send a transaction with the password that decrupts your password.

[H-2] `PasswordStore::setPassword` has no access control,which can make non-owner to set the password of the contract

Description: A non-owner can set the password of the `PasswordStore` contract by calling the `PasswordStore::setPassword` function. This is because the `PasswordStore::`

`setPassword` function does not have a modifier or a logic to check if the caller is the owner of the contract.

```
1    function setPassword(string memory newPassword) external {
2 @>       s_password = newPassword;
3          emit SetNetPassword();
4      }
```

Impact: An user can set the password of the `PasswordStore` contract, which is not the intended functionality of the contract.

Proof of Concept:

Code details here

1. Create a locall running chain

```
1 make anvil
```

2. Deploy the contract `PasswordStore`.

```
1 make deploy
```

This returns a deployed `PasswordStore` address which in our case was `0x5FbDB2315678afecb367f032d93F6`

3. Set the password of the `PasswordStore` contract by calling the `PasswordStore::setPassword` function with a random private key apart from the default private key in the make-File. We selected `0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a3492` as our private key.

```
1 cast send 0x5FbDB2315678afecb367f032d93F642f64180aa3 "setPassword(
     string  newPassword)" "changePassword"  --private-key 0
     x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a
```

This transaction does not revert and the password of the `PasswordStore` contract is set to `changePassword`.

Recommended Mitigation: We recommend to either 1. Use OpenZeppelin Ownable contracts to the contract. This will allow you to add an only owner modifier to the `PasswordStore::setPassword` function to check if the caller is the owner of the contract. 2. you can create your own modifier to the `PasswordStore::setPassword` function to check if the caller is the owner of the contract. This can be done by adding a modifier like `onlyOwner` to the `PasswordStore::setPassword` function. The `onlyOwner` modifier can be implemented as follows:

```
1 modifier onlyOwner(){
```

```
2       if (msg.sender != s_owner){
3           revert("Only owner can call this function");
4       }
5   }
```

Add the `onlyOwner` modifier to the `PasswordStore::setPassword` function as follows:

```
1   function setPassword(string memory newPassword) external onlyOwner {
2           s_password = newPassword;
3           emit SetNetPassword();
4       }
```

Informational

[I-1] The nat spec document on `PasswordStore:getPassword` function indicates a parameter that does not exist, causing the natspec to be incorrect

Description:

```
1       /**
2        * @dev Get the password of the contract
3   -->   * @param password The password of the contract
4        */
5       function getPassword() external view returns (string memory
           password) {
6           return s_password;
7       }
```

The `PasswordStore::getPassword` function does not take any parameter, but the nat spec document indicates that the function takes a parameter `password`. This causes the nat spec document to be incorrect.

Impact: The nat spec document is incorrect and can cause confusion to the users of the contract.

Recommended Mitigation: Remove the nat spec document on the `PasswordStore::getPassword` function.

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