PasswordStore Initial Audit Report

Version 0.1

Md Asif Ahamed

January 27, 2024

PasswordStore Audit Report

MD ASIF AHAMED

Januaury 26, 2024

PasswordStore Audit Report

Prepared by: MD ASIF AHAMED Lead Auditors:

- MD ASIF AHAMED

Assisting Auditors:

- None

Table of contents

See table

- PasswordStore Audit Report
- Table of contents
- About YOUR_NAME_HERE
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
- Protocol Summary
  - Roles
- Executive Summary

    - Issues found
- Findings
    - High
        * [H-1] Passwords stored on-chain are visable to anyone, not matter solidity variable visibility
        * [H-2] `PasswordStore::setPassword` is callable by anyone
- Low Risk Findings
    - L-01. Initialization Timeframe Vulnerability
        * Relevant GitHub Links
    - Summary
    - Vulnerability Details
    - Impact
    - Tools Used
    - Recommendations
        * [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

About MD ASIF AHAMED

Disclaimer

I, Md Asif Ahamed made 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 meis 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

		Impact		
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] Storing Password On-Chain Is Visiable To Public,Solidity Kewords Does Not Matters.

Description: Data Stored On_Chain Are Visibile To Anyone. In `PasswordStore::s_password` varibale that is indetened to store password and to to retrive from `PasswordStore::getPassword()` function is not recommened implementatsion as state is visiable ot anyone however the private keword is used in solidity. The Private keyword in solidity refers to stop accessing data from other contract.

Storing Sensetive Data On-Chain Is Not Recommended.

Impact:

Publically Availbale Of Sensetetive Data Like (Password)

Proof of Concept: The Test Below Show Any One retrive password Stored On s_password variable.

```
1. first Run Local Chain
```

type from the cli `anvil` and hit enter.

```
2. Deploy the contract from the deploy script
```

type from the cli `forge script script/DeployPasswordStore.s.sol:DeployPasswordStore --rpc-url http://127.0.0.1:8545 --private-key(grab a private key from the anvil network)--broadcast` hit enter and grab the contract address

```
3. Use cast with storage,contractadress, and storage which in this case
    1
```

type from the cli `cast storage 0x7ef8E99980Da5bcEDcF7C10f41E55f759F6A174B 1 --rpc-url http://127.0.0.1:8545` and hit enter.

```
4. Convert the hex form of bytes has returned 0
   x6d7950617373776f726400000000000000000000000000000000000000000014
```

type from the cli `cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000014` and hit enter.

```
5. And You Will Get the Password Stored From The Deploy Script which
   was ##myPassword
```

Recommended Mitigation: Due, to this bug the articture of the contract shoudl be rethink we don't advise to store sensetive data on-chain. The way can rethink that an encryted from of password can stored on-cahin the encryption of password can be done off-chain the the encryted data can be stored on chain. for retrival the user can get encrypted form of the of the password from the contract and then can be decrypt in of-chain cumputation.

[H-2] setPassword is missing access control, meaning anyone can store/change password

Description: `PasswordStore::setPassword()` function which is external fucntion is intendent to `This function allows only the owner to set a new password.` But it lacks the access control which is not intended and makes it to store password for anyone.

```
    function setPassword(string memory newPassword) external {
@>      //@audit no Access Control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone Can Store/Change Password

Proof of Concept: Add The Following Code To `PasswordStore.t.sol` file and then from cli type `forge test --match-test test_anyOneStorePassword` and hit enter

Code

```
      function test_anyOneStorePassword(address randomUser) public {
        vm.assume(randomUser != owner);
        vm.prank(randomUser);

        string memory newPassword = "newPassword";
```

```
6
7           passwordStore.setPassword(newPassword);
8
9           vm.prank(owner);
10
11          string memory storedPassowrd = passwordStore.getPassword();
12
13          assertEq(newPassword, storedPassowrd);
14      }
```

Recommended Mitigation: To Prenet From Storing Password Except Then The Owner Add The Following Lines of Code To The `PasswordStore.sol` file to the `setPassword` function

```
1       if(msg.sender != owner){
2           revert PasswordStore__NotOnwer();
3
4       }
```

Low Risk Findings

L-01. Initialization Timeframe Vulnerability

Submitted by dianivanov.

Relevant GitHub Links

https://github.com/Cyfrin/2023-10-PasswordStore/blob/main/src/PasswordStore.sol

Summary

The PasswordStore contract exhibits an initialization timeframe vulnerability. This means that there is a period between contract deployment and the explicit call to setPassword during which the password remains in its default state. It's essential to note that even after addressing this issue, the password's public visibility on the blockchain cannot be entirely mitigated, as blockchain data is inherently public as already stated in the "Storing password in blockchain" vulnerability.

Vulnerability Details

The contract does not set the password during its construction (in the constructor). As a result, when the contract is initially deployed, the password remains uninitialized, taking on the default value for a

string, which is an empty string.

During this initialization timeframe, the contract's password is effectively empty and can be considered a security gap.

Impact

The impact of this vulnerability is that during the initialization timeframe, the contract's password is left empty, potentially exposing the contract to unauthorized access or unintended behavior.

Tools Used

No tools used. It was discovered through manual inspection of the contract.

Recommendations

To mitigate the initialization timeframe vulnerability, consider setting a password value during the contract's deployment (in the constructor). This initial value can be passed in the constructor parameters.

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
1      /*
2       * @notice This allows only the owner to retrieve the password.
3   @>  * @param newPassword The new password to set.
4       */
5      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.

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