PasswordStore Initial Audit Report

Version 0.1

Philip Nguyen

December 16, 2023

PasswordStore Audit Report

Philip Nguyen

December 15, 2023

PasswordStore Audit Report

Prepared by: Philip Nguyen

Lead Auditors:

- Philip Nguyen

Assisting Auditors:

- None

Table of contents

See table

About Philip Nguyen

I am a Software Engineer with interest in Web3 concepts. Below is my security audit of the smart contract PasswordStore which is guided by the lead instructor of Cyfrin, Patrick Collins.

Disclaimer

I 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 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

		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	0
Info	2
Gas Optimizations	0
Total	4

Findings

High

[H-1] Passwords stored on-chain are visible to all, no matter the Solidity variable visibility.

Description: All data stored on-chain are visible and accessible to all. `PasswordStore::s_password` is intended to be a private variable, only accessible by the owner who set it through the `PasswordStore::getPassword` function.

However, anyone can access the password through various chain methodologies.

Impact: The password stored with this protocol is not private.

Proof of Concept: Here is one way a non-owner can access the password set by the contract owner.

1. Initiate an instance of Forge Anvil to mimic the blockchain environment.

```
make anvil
```

2. On a different terminal, deploy PasswordStore contract on our anvil.

```
make deploy
```

3. Copy the deployed contract address and check the variable storage for password using the following command.

We use 1 as it corresponds to the storage slot of `PasswordStore::s_password`.

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

We will get an output that looks like this:

0x6d7950617373776f726400000000000000000000000000000000000000000014

4. Parse the returned hex value.

```
    cast parse-bytes32-string <hashed_password>
```

The returned value is our PasswordStore::s_password:

myPassword

Recommended Mitigation: Due to this vulnerability, the overall architecture of this protocol must be rethought. A viable alternative is encrypting the password off-chain, then storing the encrypted password on-chain.

A disadvantage of this alternative method is that it would require the user to remember another password to decrypt their encrypted password.

Please note that if you take this approach, it is recommended to remove the view function so that the user does not accidentally send a transaction with the password that decrypts their password.

[H-2] PasswordStore::setPassword has not access control, anyone can change the password.

Description: The natspec describes the overall purpose of PasswordStore::setPassword to be: This function allows only the owner to set a **new** password.

However, the absence of access control enables nearly anyone to visit this contract and set the password with the PasswordStore::setPassword function.

```
    function setPassword(string memory newPassword) external {
@>      // @audit-h There are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password, which severely impacts the intended purpose of the protocol.

Proof of Concept: Add the following test to the PasswordStore test suite.

Code

```
``` javascript
        function test_non_owner_can_set_password(address randomAddress)
            public {
```
```

```
            // assume that the randomAddress cannot be the owner
            vm.assume(randomAddress != owner);

            // generate random address that is not the contract owner
            vm.prank(randomAddress);
            string memory expectedPassword = "myNewPassword";
            passwordStore.setPassword(expectedPassword);

            // retrieve the password to see if our password has been
               set by non-owner
            vm.prank(owner);
            string memory actualPassword = passwordStore.getPassword();
            assertEq(actualPassword, expectedPassword);
        }
    ```
```

If this test passes, we know that a non-owner can set the password.

**Recommended Mitigation:** Add access controls to `PasswordStore::setPassword` function.

Recommended Code Changes

```
``` javascript
        function setPassword(string memory newPassword) external {
    +       if (msg.sender != s_owner) {
    +           revert PasswordStore__NotOwner();
    +       }
            s_password = newPassword;
            emit SetNetPassword();
        }

    ```
```

Informational

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

Description: The `PasswordStore::getPassword` function signature is `getPassword()` but its 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.
```
```

[I-2] Event `PasswordStore::SetNetPassword` should be named `PasswordStore::SetNewPassword`, may be confusing due to users

Description: The event `PasswordStore::SetNetPassword` has a typo. The event may be more accurately represented if renamed to `PasswordStore::SetNewPassword`.

This change enhances readability and reduces confusion.

Impact: `PasswordStore::SetNetPassword` presents a typo which may confuse users, developers, and others reading the contract.

Recommended Mitigation: Rename `PasswordStore::SetNetPassword` to `PasswordStore::SetNewPassword`

```
1
2 -     event SetNetPassword();
3 +     event SetNewPassword();
```