PasswordStore Audit Report

Version 1.0

Manish Roy

December 14, 2023

Prepared by: Manish Roy

Lead Security Researcher: - Manish Roy

Index

Table of Contents

Disclaimer

I, the security researcher, have made 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 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.

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.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

The CodeHawks severity matrix has been leveraged to determine severity of the bugs. See the documentation for more details.

Audit Details

The findings described in this document correspond to the following commit hash:

`2e8f81e263b3a9d18fab4fb5c46805ffc10a9990`

in the following repository:

https://github.com/Cyfrin/2023-10-PasswordStore

Scope

```
src/
|--- PasswordStore.sol
```

Roles

- Owner: the only one who should be able to set and access the password, for this contract.

Issues found

Severity	No. of Issues
High	2
Medium	0
Low	1
Informational	1
Gas	0
Total	**4**

Findings

High

[H-01] Passwords stored on-chain are visable to anyone, irrespective of solidity variable visibility

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable, and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

However, anyone can directly read this using any number of off chain methodologies

Impact: The password is not private.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

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 that's the storage slot of `s_password` in the contract.

   ```
   cast storage {CONTRACT_ADDRESS} 1 –rpc-url {RPC_URL}
   ```

   You'll get an output that looks like this:

   `0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. *(Optional)* You can then parse that hex to a string with:

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

   And get an output of:

   `myPassword`

Recommended Mitigation: It is advisable to reconsider the contract's overall architecture. One suggestion is to encrypt the password outside of the blockchain and then store the encrypted version on the blockchain. This would require the user to remember another password to decrypt the password. Furthermore, it would be a good idea to remove the function that allows others to view the password, as we wouldn't want the user to accidentally send the password along with their transaction.

[H-02] `PasswordStore::setPassword` is callable by anyone

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 the owner to set a new password.`

```
1      function setPassword(string memory newPassword) external {
2  ~>      // @audit - There are no access controls here
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: Anyone can set/change the password of the contract.

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

```
1     function test_anyone_can_set_password(address randomAddress) public
          {
2         vm.assume(randomAddress != owner);
3
4         vm.prank(randomAddress);
5         string memory expectedPassword = "myNewPassword";
6         passwordStore.setPassword(expectedPassword);
7
8         vm.prank(owner);
9         string memory actualPassword = passwordStore.getPassword();
10        assertEq(actualPassword, expectedPassword);
11    }
```

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

```
1     if (msg.sender != s_owner) {
2         revert PasswordStore__NotOwner();
3     }
```

Low

[L-01] Initialization timeframe vulnerability

Description: There is a period between contract deployment and the explicit call to `PasswordStore::setPassword` function 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.

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.

Recommended Mitigation: 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.

Informational

[I-01] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description: 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()`.

```
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) {
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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