Protocol Audit Report

Version 1.0

heheboii.eth

December 24, 2024

PasswordStore-Audit

Karthik Reddy

Dec 15, 2024

Prepared by: Karthik Reddy (https://github.com/heheboii11)

Lead Auditors: - Karthik Reddy

Table of Contents

Protocol Summary

Password store is contract that stores password and only the owner can set and view the password from the contract when required. This protocol is designed for a single person and cannot accomidate multiple persons.

Disclaimer

The Aduitor Karthik 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 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 corresponds to the following commit hash:

Commit Hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
In Scope:
./src/
#  PasswordStore.sol
```

Roles

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

Executive Summary

We have found 2 highs in this report and a info finding in this report, even though this is a inspiration and basic audit from cyfrin updraft security course, I promise myself O will gradually increase my security findings and want to contribute towards the secured web3 ecosystem

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 Data On-Chain Makes It Visible to Anyone; Data Is No Longer Private

Description: All the data stored on the blockchain is visible to anyone and can be accessed by anyone. Storing the variable as `private` does not make it invisible. The variable `Passwordstore::s_password` is intended to be readable only by the owner. To achieve this, an off-chain mechanism is required to handle the data securely.

Impact: Anyone can read the private variable from the blockchain.

Proof of Concept: You can retrieve the data from the blockchain as follows:

1. **Create a local running chain:**

   ```
   make anvil
   ```

2. **Deploy the contract to the chain:**

   ```
   make deploy
   ```

3. **Run the cast command to retrieve the storage data:**

   ```
   cast storage <Address of the contract> 1 --rpc-url http
       ://127.0.0.1:8545
   ```

   Replace `<Address of the contract>` with the deployed contract's address. The `1` is used because the variable `s_password` occupies storage slot `1`, while `s_owner` uses slot `0`.

4. **Convert the retrieved bytes32 data to a string:**

   ```
   cast parse-bytes32-string <bytes32 data retrieved>
   ```

Recommended Mitigation: The architecture of the code must be changed. Do not store passwords in a private variable on-chain. Instead, use an off-chain method to securely store and access the password when the owner requires it. While encryption is an option, it requires remembering an additional password for decryption.

[H-2] No Restriction on `PasswordStore::setPassword` Allowing Anyone to Change the Password

Description: There is no restriction on who can set or change the password, allowing anyone to modify it. A restriction should be enforced to ensure only the owner can change or set the password.

Impact: Anyone can set or change the password, which is a critical vulnerability.

Proof of Concept: In the test file, add the following code to demonstrate that anyone can set the password:

Code

```
function testNotOwnerCanSetPassword(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.startPrank(randomAddress);
    string memory newPassword = "MyNewPassword";
    passwordStore.setPassword(newPassword);
    vm.stopPrank();

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();

    assertEq(newPassword, actualPassword);
}
```

Recommended Mitigation: Add an `if` statement or a modifier to ensure only the owner can set the password.

Code

Using an `if` statement:

```
if (msg.sender != owner) {
    revert PasswordStore__NotOwner();
}
```

Using a modifier for robustness:

```
modifier onlyOwner() {
    if (msg.sender != owner) {
        revert PasswordStore__NotOwner();
    }
    _;
}
```

Low

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.

Informational

[I-1] Unused Parameter newPassword in NatSpec for `PasswordStore::getPassword`

Description: The parameter `newPassword` is present in the NatSpec for the function `PasswordStore::getPassword` but is unused and unnecessary. This should be removed.

Impact: The NatSpec is incorrect, leading to potential confusion.

Recommended Mitigation: Remove the incorrect NatSpec line:

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