PasswordStore Protocol Audit Report

Version 1.0

Cyfrin.io

December 25, 2023

PasswordStore Protocol Audit Report

0xJustUzair

December 24, 2023

Prepared by: 0xJustUzair Lead Security Researcher:

- 0xJustUzair

Table of Contents

Protocol Summary

The `PasswordStore` contract assumes that only the owner can set the password. The `setPassword()` function modifies the `s_password` storage variable, where the password is set, but doesn't include access control meaning that anyone, including a malicious actor, can reset the owner's password.

Disclaimer

The `0xJustUzair` team 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

Commit Hash : `7d55682`

Scope

- Commit Hash: 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

- In Scope:

```
./src/
#-- PasswordStore.sol
```

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

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

Spent 30 mins auditing the protocol solo, with foundry and other built-in tools

Issues found

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

Findings

High

[H-1] Password stored on-chain makes it visible to anyone, and no longer private

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.

We show one such method of reading any data off chain below.

Impact: Anyone can read the 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 locally running chain

```
make anvil
```

2. Deploy contract on chain

```
make deploy
```

3. Run the storage tool

```
cast storage <DEPLOYED_CONTRACT_ADDRESS> 1 --rpc-url http://localhost
    :8545
```

You get output as such: `0x6d7950617373776f72640000000000000000000000000000000000000000001`

4. Convert the output to a readable string

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

You get output as such: `myPassword`

Recommended Mitigation:

1. encrypt the password off chain and then store encrypted password on chain
2. User would reqiure to remmeber another password for decrpytion of encrypted password
3. Remove view function, as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password

[H-2] `PasswordStore::setPassword()` has no access controls, meaning a non-owner could change the password

Description: `PasswordStore::setPassword()` function is set to be an external function, the natspec of the function and overall purpose of the smart contract is that `The function allows only owner to set a new password`

```
function setPassword(string memory newPassword) external {
@>      // @audit - missing access control
        s_password = newPassword;
        emit SetNetPassword();
}
```

Impact: Anyone can set the password of the contract breaking the functionality of the contract.

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

Code

```
function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: Add an access control conditional to `setPassword()` function.

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

Informational

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

Description: `PasswordStore::getPassword()` natspec indicates signature `PasswordStore::getPassword(string)` while actual code indicates `PasswordStore::getPassword()`

```
 @param newPassword The new password to set.
```

Impact: The natspec is incorrect

Recommended Mitigation: Remove incorrect natspec

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