Protocol Audit Report

Version 1.0

Cyfrin.io

July 3, 2024

Protocol Audit Report

Ghislain Te Selone

July 1, 2024

Prepared by: [Ghislain Te selone] Smart Contract Lead Security Researcher

Table of Contents

Protocol Summary

A smart contract application for storing a password. Owner should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

We make 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 correspond to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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

Add some notes about how the audit went, types of things you find, etc. We spent X hours with Z auditors using Y tools etc.

Issues found

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

Findings

High

[H-1] STORAGE VARIABLE `PasswordStore::s_password` can be retrieved by anyone even if marked as PRIVATE, meaning that password stored is not safe

Description: While storing password inside `PasswordStore::s_password` may seem safe due to its private visibility, it's actually not. Anybody can still access the password, as will be shown below in the proof of concept. Visibility can only protect you from function callers.

**Impact:* Password safety can't be guaranteed

Proof of Concept: Here is how we can retrieve any password stored in `PasswordStore::s_password` even if we are not the owner and no matter what the visibility is:

1. Open the terminal and run a local chain (Anvil) :

```
anvil
```

2. Open a second terminal and run the following to deploy PasswordStore contract on anvil (don't close the first terminal opened because it will stop anvil) :

```
make deploy
```

3. Copy the contract address once deployed and run the following :

```
cast storage <contract address> 1
```

4. Copy the bytes coming out pf the precedent run and run this :

```
cast parse-bytes32-string <bytes from precedent run>
```

As result you will get "myPassword" as expected.

Recommended Mitigation: The best advice i can provide is to find a way to encrypt the password off-chain and store the encrypted password on chain. That way even if a IQ2500 brain crack it , he would'nt be able to know right off the bat what the actual password is.

[H-2] Lack of access controls on `PasswordStore::setPassword` function, meaning that password integrity is not safe

Description: The `PasswordStore::setPassword` function not having an access control can be dangerous for the protocol as it let us know right inside the natspec that owner should be the only user able to call it successfully. In the current state of the function anyone can set or modify a password previously set by the owner.

Code

```
function setPassword(string memory newPassword) external {
        // An access control should be set here
        // Revert when user is not allow
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Password integrity cant be guarranted by the protocol.

Proof of Concept: To prove it we are going to write a new test function called `test_any_user_can_call_it` inside the `PasswordStoreTest` contract where the test contract (`PasswordStoreTest`)

address itself will be able to set a new password, different from the one set when deploying the `PasswordStore` contract.

Code

```
function test_any_one_can_call_it() external {
        // Check that the password set when deploying the contract
        // ... is "myPassword"
        vm.prank(owner);
        string memory previousPass = passwordStore.getPassword();

        assertEq(
            keccak256(abi.encodePacked(previousPass)),
            keccak256(abi.encodePacked("myPassword"))
        );

        // Set the new password by this contract address
        passwordStore.setPassword("selone");

        // Retrive the new password
        vm.prank(owner);
        string memory newPass = passwordStore.getPassword();

        // Check if it changes to "selone"
        assertEq(
            keccak256(abi.encodePacked(newPass)),
            keccak256(abi.encodePacked("selone"))
        );
    }
```

Recommended Mitigation: Add a `onlyOwner` like modifier on the function or a direct line of code implementing the exclusion of non owner user from calling `PasswordStore::setPassword` function.

```
+  if(msg.sender != s_owner) revert PasswordStore__OnlyOwner();
```

Medium

Low

Informational

[I-1] Uneccessary @param newPassword expected inside `PasswordStore::getPassword` function meaning that it serves no purpose

Description: The `PasswordStore::getPassword` function natspec mention a @param newPassword to be implemented inside the function but in reality that @param serve no purpose as the function itself already perform correctly what it is supposed to do.

Impact: No impact as it is uneccessary.

Recommended Mitigation: Suppress the line where @param newPaswword is mentioned as it has nothing to do there.

Check element in cause

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