Protocol Audit Report

Version 1.0

Cyfrin.io

January 7, 2024

Protocol Audit Report

Andi Putra

Jan 6, 2024

Prepared by: Andi Putra Lead Auditors:

Andi Putra

Table of Contents

Protocol Summary

PasswordStore is created for an owner to be able to set a password and retrieve it. Other users should not be able to retrieve the user's stored password.

Disclaimer

Andi Putra 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

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

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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 spent X hours with Z auditors using Y tools and found X issues.

Issues found

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

Findings

High

[H-1] Storing the password 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, to be accessed only through `PasswordStore::getPassword` function. The `PasswordStore::getPassword` function is intended to be successfully called only by the owner of the contract.

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

Impact: Anyone can read the private password, severely breaking the functionality of the protocol.

Proof of Concept:

The below test case shows how anyone can reaed the password directly from the blockchain.

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 is the storage slot for `s_password` in the contract.

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

You'll get an output that look like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse the hex to a string with:

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

And, get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be reconsidered. One could encrypt the password off-chain and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the `view` function as you wouldn't want to accidentally send a transaction with the password that decrypt your password.

[H-2] `PasswordStore::setPassword` has no access control. Meaning, a non-owner can change the password.

Description: There is no check for owner in `PasswordStore::setPassword`, which is an `external` function. According to the natspec of the function, `This function allows only the owner to set a new password.`

```
function setPassword(string memory newPassword) external {
=>  // @audit: There is no access control
    s_password = newPassword;
    emit SetNetPassword();
}
```

Impact: Anyone can change the password set by the owner, thus severely breaking the contract intended functionality.

Proof of Concept:

Add the following to `PasswordStore.t.sol` test file. And run test.

```
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(expectedPassword, actualPassword);
}
```

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

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

Informational

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

Description:

```
/*
  * @notice This allows only the owner to retrieve the password.
  * @param newPassword The new password to set.
  */
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    return s_password;
}
```

The `PasswordStore::getPassword` natspec indicates a `newPassword` parameter. But the function signature `getPassword()` does not have any parameter.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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