Protocol Audit Report

Version 1.0

Cyfrin.io

May 2, 2024

Protocol Audit Report

Damboy.eth

May 1, 2024

Prepared by: Damboy.eth Lead Security Researcher: - Me

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - The findings are described in the document corresponding the following commit hash:**
  - Scope
  - Roles
- Executive Summary
  - Issues found
  - Findings
  - High
    * [H-1] Storing Password on-chain makes it visible to anyone, and no longer private
    * [H-2] TITLE `PasswordStore::setPassword` has no access control, meaning anyone that is not an owner could change the password
  - Informational
    * [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist, causing the natspec to be incorrect.

Protocol Summary

PasswordStore is a protocol dediated for storage and retrival of a user's password. The protocol is designed to be used by a single user, and it is not designed to be used by multiple users. Only the owner should be able to access this password.

Disclaimer

The YOUR_NAME_HERE 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

The findings are described in the document corresponding the following commit hash:**

```
hash
```

Scope

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

Roles

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

Executive Summary

Issues found

Severity	No of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

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

Description: All data stored on-chain can be 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 private password, severely breaking the functionality of the protocol

Proof of Concept:(Proof of code)

The below test shows how anyone can read 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 of `s_password` in the contract.

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

you'll see something like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

you can then pass that 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 this contract should be rethought. One could encrypt 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 likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] TITLE `PasswordStore::setPassword` has no access control, meaning anyone that is not an owner could change the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and the 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 is not access control
```

```
3           s_password = newPassword;
4           emit SetNetPassword();
5       }
```

Impact: Anyone can set/change the password of the contract, severely breaking the contract intended functionality.

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

Code

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

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

```
1  if(msg.sender != s_owner)
2  {
3      revert PasswordStore__NotOwner();
4  }
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist, causing the natspec to be incorrect.

Description:

```
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) {
```

The `PasswordStore::getPassword` fuction signature is `getPassword()` which the natspec says it should be `getPassword(string)` .

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

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