AERARIUM

Protocol Audit Report

Version 1.0

Cyfrin.io

May 5, 2024

Protocol Audit Report

Aerarium.io

May 7, 2023

Prepared by: Aerarium Lead Auditors: - Terome J. Mensah, Lead Security Researcher

Table of Contents

    * [I-1] TITLE (Root Cause + Impact)
  - `PasswordStore::getPassword` natspec identifies a parameter that doesn't exist, causing the natspec to be incorrect.
  - Gas

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Aerarium 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 described in the following document correspond to the following commit hash: **

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

- 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.
- Outsiders: No one else should be able to set or read the password.

Executive Summary

- Add notes about how the audit went. The types of issues you found, etc. *
- E.g. We spent X hours with Z auditors, using Y tools, etc. *

Issues found

Severity	Number of Issues
High	2
Medium	0
Low	0
Info	2
Total	3

Findings

High

[H-1] TITLE (Root Cause + Impact)

Impact H Likelihood H Severity H ## E.G., Storing the password on-chain makes it visible to anyone and no longer private.

Description: All data stored on the blockchain 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 `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 case shows how anyone could read the password directly from the blockchain. We use Foundry's `cast` tool to read directly from the storage of the contract, without being the owner.

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's the storage slot of `s_password` in the contract.

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

   You'll get an output that looks like this:

   ```
   0x6d7950617373776f726400000000000000000000000000000000000000000014
   ```

   You can then parse 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 the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password on-chain to decrypt the password. However, you'd also 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 (Root Cause + Impact)

Impact - H Likelihood - H Severity - H **Title:** PasswordStore::setPassword is callable by anyone

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

```
function setPassword(string memory newPassword) external {
    // @audit - There are no access controls here
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact: Anyone can set/change the password of the contract.

Proof of Concept:

Code

```
function test_anyone_can_set_password(address randomAddress) public {
    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 modifier to the `setPassword` function.

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

Medium

Low

Informational

[I-1] TITLE (Root Cause + Impact)

Impact - None Likelihood - High Severity - Informational / Gas / Non-Crit

`PasswordStore::getPassword` natspec identifies 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)
```

The `PasswordStore::getPassword` function signature is `getPassword()`, which the namespace says should be `getPassword(string)`."

Impact:

The natspec is incorrect.

Recommended Mitigation:

Remove incorrect natspec line.

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

Gas