Protocol Audit Report

Version 1.0

Cyfrin.io

Protocol Audit Report

Version 1.0

Cyfrin.io

May 2, 2024

Protocol Audit Report

Seven Cedars

May 1, 2024

Prepared by: Seven Cedars Lead Auditors: Seven Cedars - xxxxxxx

Table of Contents

Protocol Summary

The protocol does X, Y, Z…

Disclaimer

The Seven Cedars 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
Likelihood	High	H	H/M	M
	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

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

Severity	Number of Issues found
high	2
medium	0
low	0
info	1
total	3

Issues found

Findings

High

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

Description: All on-chain data is visible to anyone, as it can be read directly from blockchain. The `PasswordStore::s_password` variable is intended to be private and only callable through the `PasswordStore::getPassword` function. Instead, s_passowrd is stored on_chain and hence publicly accesible.

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

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

Proof of Concept: The below test case shows one method of reading `PasswordStore::s_password` of the blockchain.

1. create a locally running chain

```
make anvil
```

2. deploy contract to the chain

```
make deploy
```

3. run storage tool

```
cast storage 0x5fbdb2315678afecb367f032d93f642f64180aa3 1
```

4. parse bytes32 storage to string

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

Output: `myPassword`

Recommended Mitigation: The overall architecture needs to be rethought. Storage of any password (and its encoding) need to happen off-chain.

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

Description: The `PasswordStore::setPassword` lacks any access control. It means that any address can call the function and change the password. This is contrary to the natspec that states that `This function allows only the owner to set a new password`.

```
    function setPassword(string memory newPassword) external {
    @>   //@audit no access control in previous line.
        s_password = newPassword;
        emit SetNetPassword();
    }
```

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

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

expand 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 the `setPassword` function.

expand code

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

Informational

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