Protocol Audit Report

Version 1.0

Nihavent

January 15, 2024

Password Store Audit Report

Nihavent

Jan 14, 2024

Prepared by: [Nihavent] Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

Protocol does X, Y, Z

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 in this document correspond to the follwoing Commit Hash:

```
xxx
```

Scope

```
./src/
```

```
-- PasswordStore.sol
```

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

Issues found

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

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone

Description

All data stored on-chain is visible to anyone, and can be read directly from the blockcahin. The `PasswordStore::s_password` variable is intended to only be visible to the owner of the contract through the `PasswordStore::getPassword` function.

Impact

Anyone can read the private password, severly breaking the functionaility of the protocol.

Proof of Concept (proof of code)

The below test case 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's the storage slot of `PasswordStores_password` in the contract.

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

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

Parse the hex to a string:

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

And get an output of:

`myPassword`

Recommended Mitigation

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 off-chain to decrypt the password. This would present a new risk such as the user accidently sending a transaction with the password that decrypts the actual password, instead of their new desired password.

[H-2] TITLE `PasswordStore::s_password` has no access controls, meaning a non-owner could change the password.

Description

The `PasswordStore::s_password` function is set to be an `external` function, however, the natspec of the function and overall purpose of the smart contract says that `This function allows only the owner to set a new password.`

Impact

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

Anyone can set/change the password of trhe contract, breaking the intended functionality of the control.

Proof of Concept (proof of code)

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

Code

```
    function testAnyoneCanSetPassword(address randomAddress) public {
        vm.assume(randomAddress != owner);

        // Setting password as a random address
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

        // Checking password as the owner
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation

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

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

[I-1] TITLE The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist.

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()` while the natspec says it should be `getPassword(string newPassword)`

Impact

The natspec is incorrect.

Proof of Concept (proof of code)

Recommended Mitigation Remove the incorrect natspec line.

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