Preet's Security Reviews

Securing the Future of Blockchain

Protocol Audit Report

Version 1.0

Preet Singh

June 10, 2024

Protocol Audit Report

Preet Singh

June 10, 2024

Prepared by: Preet Singh Lead Security Researchers:

- Preet Singh

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of the user's password. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access the password.

Disclaimer

The Preet Singh 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 this document corresponds 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

- I spent almost 5 hours in auditing this smart contract. I have used a tool called cloc and I have founded total three issues in your code. Briefly decribed below.

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 is 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 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 Concepts (Proof of Code)

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 the storage slot of `s_password` in the contract.

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

you will get an output look like this: `0x6d7950617373776f726400000000000000000000000000`

you can then parse that hex to a string with :

```
cast parse-bytes-32-string 0
    x6d7950617373776f7264000000000000000000000000000000000000000000000014
```

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 off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidently send the transaction with the password that decrypts the password.

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

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

```

    function setPassword(string memory newPassword) external {
->      // @audit any user can set a Password
        s_password = newPassword;
        emit SetNetPassword();
    }
```

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

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

Code

```
    //Testing that anybody can set Password
    function test_anybody_can_set_password(address randomAddress)
        public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "My New Password";
        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.

```

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

Informational

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

Description

```

  // @audit There is no parameter for getPassword function
    function getPassword() external view returns (string memory) {
```

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

Impact The natspec is incorrect.

Recommended mitigation Remove the incorrect natspec line.

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