Protocol Audit Report

Version 1.0

Akshat

July 9, 2024

Protocol Audit Report

Akshat

July 9 ,2024

Prepared by: Akshat Lead Security Researcher: - Akshat

Table of Contents

Protocol Summary

Protocol stores the password and allows user to read it.

Disclaimer

The Akshat 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

The findings described in this document correspond with the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

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

Executive Summary

describe your experience , what you found , etc.

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 , no longer private

Description: All the data stored on-chain can be read directly from the blockchain. The `PasswordStore::s_password` variable in intended to be a private variable and accessed only through the `PasswordStore::getPassword` function , which is intended to be called by only 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 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 `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 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 accidentally send a transaction with the password that decrypts your password.

Likelihood & Impact

- Impact : HIGH
- Likelihood : HIGH
- Severity : HIGH

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

Description: The `PasswordStore::setPassword` function is set to be external, however the natspec and the 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
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password, severely breaking the contract's intended functionality

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

Code

```
    function testAnyoneCanSetPassword(address randomAddr) public
        {
            vm.assume(randomAddr!=owner);
            vm.prank(randomAddr);
            string memory newPass = "hehe i am hecker";
            passwordStore.setPassword(newPass);

            vm.prank(owner);
            string memory actualPass = passwordStore.getPassword();

            // assert(actualPass == newPass);
            assertEq(actualPass,newPass);
        }
```

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

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

Likelihood & Impact

- Impact : HIGH
- Likelihood : HIGH
- Severity : HIGH

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesnt 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 natspec indicates should be `getPassword(string)`

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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

Likelihood & Impact

- Impact : NONE
- Likelihood : NONE
- Severity : Informational/Gas/non-crits