Protocol Audit Report

Version 1.0

equious.eth

June 8, 2024

Protocol Audit Report

edumelo

Jun 8, 2024

Prepared by: edumelo Lead Security Researcher: - edumelo

Table of Contents

Protocol Summary

The protocol stores a password set by the owner in the block

Disclaimer

The edumelo 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 correspond the following
   commit hash **
```
    0xd28EB68b2ad007209D08e1214cb6EcB3d7b097Ee
```
```

Scope

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

Roles

- Owner: the user who can set the password and read the password.
- Outside: 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, and no longer private

Description: All data stored on-chain in 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 the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

Impact: Anyone can read the private password, severly 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 stot 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 the parse that hex to a string width:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: Due this, the overall architecture of the contract should be rethough. One could encrypt the password off-chain, and the 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 contreols, 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 the `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 of the contract, severly breaking the contract intended functionality.

Proof of Concept:

Code

```
    function test_anyone_can_set_password(address randomAddress) public
        {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "myPassword";
        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();
    }
```

Likelihood & Impact

- Impact: HIGH

- Likelihood: HIGH
- Severity: HIGH

Medium

Low

Informational

[I-1] The PasswordStore::setPassword NatSpec Indicates a Non-Existent Parameter Causing Incorrect Documentation

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::setPassword` function signature is `getPassword()` which the nastspec says it should be `getPassword(string)`

Impact: The natspec is incorrect

Recommended Mitigation:

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

Likelihood & Impact

Gas