Protocol Audit Report

Version 1.0

Norbert Orgován

January 15, 2024

Protocol Audit Report

Norbert Orgovan

January 15, 2024

Prepared by: Orgovan & Churros

Lead Auditors: - Norbert Orgován

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storing and retriving a user's pwd. The protocol is designed to be used by a single user, not multiple users. Only the owner should be able to set and access this pwd.

Disclaimer

The Orgovan & Churros 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 fundings described in this document correspond to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

Roles

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

Executive Summary

Some notes about how the audit went, types of findings, etc. We spent X hours with Y auditors, using Z tools.

Issues found

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

Findings

High

[H-1] Storing the pwd on-chain makes it 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 the `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 Concept: (Proof of Code)

The test case below shows how anyone can read the pwd 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 is the storage slot of the `PasswordStore::s_password` in the contract.

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

You will get an output like this: `0x6d7950617373776f72640000000000000000000000000000000000000000000`

You can then parse this hex to a string as follows:

```
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 pwd off-chain, and then store the encyprted pwd on-chain. This would require the user to remember another pwd off-chain to decrypt the pwd. However, you would also likely want to remove the view function as you would not want the user to accidentally send a transaction with the pwd that decrypts your pwd.

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

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

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

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

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

Code

```
    function test_anyone_can_set_password(address randomAddress) public
         {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.etPassword(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 != owner){
    revert PasswordStore__NotOwner();
}
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not 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 siganture is `getPassword()`, but the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Proof of Concept: -

Recommended Mitigation: Remove the incorrect natspec line.

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