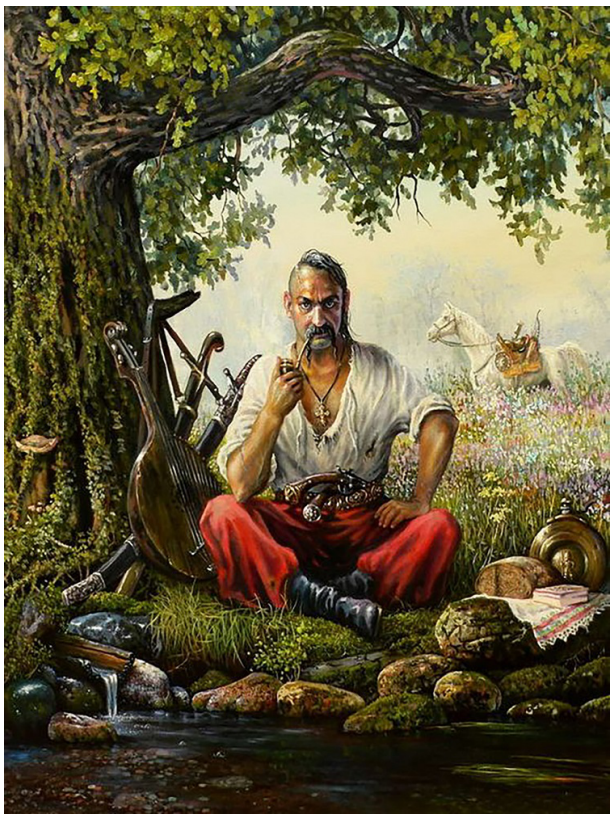

Protocol Audit Report

Version 1.0

khimcdmy

December 30, 2023

Protocol Audit Report

khimcdmy

Dec 31, 2023

Prepared by: khimcdmy Lead Security Researcher: - Dmytro Khimchenko

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. 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 this password.

Disclaimer

The khimcdmy 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: Commit Hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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 or read the password.

Executive Summary

We spent 5 hours with 1 autditor using manual testing

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 more 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 inteded 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, 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

```
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 will get an output that lookl like this: `0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architechture 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.

[H-2] Missing access controls on `PasswordStore::setPassword` makes possible setting the new password by anyone

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 the owner to set new password.` (direct quote from the codebase)

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

```
    function test_anyone_can_set_password(address randomAddress) public {
        // 100% assume
        vm.assume(randomAddress != owner);

        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        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` natspec indicates a parameter that doesn't 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.
*/


```

```
7  function getPassword() external view returns (string memory) {
8      if (msg.sender != s_owner) {
9          revert PasswordStore__NotOwner();
10     }
11     return s_password;
12 }
```

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

Impact: The natspec is incorrect

Recommended Mitigation: Remove incorrect natspec line

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