Protocol Audit Report

Version 1.0

L.Petroulakis

January 30, 2024

Protocol Audit Report

L.Petroulakis

January 29, 2024

Prepared by: L.Petroulakis Lead Security Researcher: - L.Petroulakis

Table of Contents

Protocol Summary

This is a protocol dedicated to storage and retrieval of the user's passwords. it is designed to be used by a single user, and only the owner should be able to set and read the passwords.

Disclaimer

An effort was made to find as many vulnerabilities in the code in the given time period, but no responsibilities are held for the findings provided in this document. A security audit 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 finding described in this document correspond to the following commit hash :

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
PasswordStore.sol
```

Roles

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

Executive Summary

The review was smooth and the code was easy to read. The code is well documented and the comments are clear. The code is well structured and the functions are well named. The code is well tested and the tests are easy to read and understand. The code is well formatted and the code style is consistent.

Issues found

Severity	Issues Found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing a password 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` is intended to be a private variable and only accessed though the `PasswordStore::getPassword` function, which is intended to be called by the contract's owner.

We show one such method of reading any data off chain below.

Impact: Anyone can read the private password, severely breaking the funcitonality of the contract.

Proof of Concept: (proof of code)

the test case below 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 protocol 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 on-chain. 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.

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

Description: The `PasswordStore::setPassword` function is set as an `external` function but the purpose of the smart contract is to `only allow the owner to set a new password.`

```
function setPassword(string memory newPassword) external {
@>      // @audit - no access controls, anyone can change the password
```

```
3           s_password = newPassword;
4           emit SetNetPassword();
5       }
```

Impact: Anyone can set or change the password, severely breaking the funcitonality of the contract.

Proof of Concept: add the following to the `test/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.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: Add an access control conditional to the `setPassword` function to ensure only the owner can set the password.

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

Informational

[I-1] `PasswordStore::getPassword` param indicates that a parameter should exist, but it does not, so it's 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) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

The `PasswordStore::getPassword` function sig is `getPassword()`, but the comment indicates that it should be `getPassword(string)`.

Impact: Incorrect natspec

Recommended Mitigation: Remove the incorrect natspec line.

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

Gas