Protocol Audit Report

Version 1.0

0xwolficy

February 29, 2024

Password Store Audit Report

0xwolficy

December 12, 2023

Prepared by: 0xwolficy (https://wolficy.xyz) Lead Auditors: 0xwolficy

Table of Contents

    * Impact: Anyone can read the password, severely breaking the protocol's purpose.
    * Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.
    * Recommended Mitigation: If possible, don't store sensitive data on-chain. Otherwise, the entire protocol's architecture should be rethought. Maybe encrypt the password off-chain and save it already encrypted on-chain.
  - [H-2] Missing access control in `PasswordStore::setPassword`. Anyone can set a new password.
    * Description: It's intended that only the owner can set a new password. However, the `PasswordStore::setPassword` function does not check that it's the owner who makes the call to set the password, resulting in anyone being able, owner or not, to set a new password. This has a very negative effect in the protocol's supposed privacy.
    * Impact: Anyone can set a new a password.
    * Proof of Concept: Add the following to the `PasswordStore.t.sol` file.
    * Recommended Mitigation: Check if it's the owner himself that is making the call before anything else. If it's not, revert, otherwise go ahead and set a new password. This can be easily done with the `require` built-in solidity method, or even with an `if` statement.

- Medium
- Low
- Informational
  - [I-1] The `PasswordStore::getPassword` function natspec is incorrect.
    * Description: The `PasswordStore::getPassword` function natspec is incorrect since it indicates that it takes a string parameter when, in fact, it does not.
    * Impact: The `PasswordStore::getPassword` function natspec is incorrect.
    * Recommended Mitigation: Remove the natspec line regarding the use of `newPassword` param
- Gas

Protocol Summary

Protocol let users set new passwords and change old ones, securing them privately to be seen only by their respective owners.

Disclaimer

The 0xwolficy 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

Scope

The contracts in scope were the next:

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

Roles

```
- Owners: can set and read their passwords.
- Outsiders: neither can set nor read any password.
```

Executive Summary

Issues found

Three (3) issues were find during this security review.

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

Findings

High

[H-1] Data stored on-chain is public. Private password variable could be read by anybody, breaking the protocol's entire purpose.

Description: All data that is stored on the blockchain is public and can be read by everyone. The `PasswordStore::s_password` variable, despite of the solidity syntax used with it, it's stored on-chain and therefore is public, going against the protocol's ethos of it being private. It's intended that this variable can only be accessed by the `PasswordStore::getPasword` function and only by the owner himself.

Impact: Anyone can read the password, severely breaking the protocol's purpose.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

Create a locally running chain

```
make anvil
```

Deploy the contract to the chain

```
make deploy
```

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
```

(usually we don't need to be this specific for a private audit and such an "odd" vuln, but for competitive audits verbosity works)

Recommended Mitigation: If possible, don't store sensitive data on-chain. Otherwise, the entire protocol's architecture should be rethought. Maybe encrypt the password off-chain and save it already encrypted on-chain.

[H-2] Missing access control in `PasswordStore::setPassword`. Anyone can set a new password.

Description: It's intended that only the owner can set a new password. However, the `PasswordStore::setPassword` function does not check that it's the owner who makes the call to set the password, resulting in anyone being able, owner or not, to set a new password. This has a very negative effect in the protocol's supposed privacy.

```
    function setPassword(string memory newPassword) external {
@>      //lack of access control, anyone can set a password
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set a new a password.

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

Code

```
    function test_non_owner_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory newPassword = "newPassword";
        passwordStore.setPassword(newPassword);
        vm.stopPrank();

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

Recommended Mitigation: Check if it's the owner himself that is making the call before anything else. If it's not, revert, otherwise go ahead and set a new password. This can be easily done with the `require` built-in solidity method, or even with an `if` statement.

```
    function setPassword(string memory newPassword) external {
@>      require(s_owner == msg.sender, 'Not the owner')
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Medium

No issues found.

Low

No issues found.

Informational

[I-1] The `PasswordStore::getPassword` function natspec is incorrect.

Description: The `PasswordStore::getPassword` function natspec is incorrect since it indicates that it takes a string parameter when, in fact, it does not.

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
        ...
    }
```

Impact: The `PasswordStore::getPassword` function natspec is incorrect.

Recommended Mitigation: Remove the natspec line regarding the use of `newPassword` param

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

Gas

No issues found.