Protocol Audit Report

Version 1.0

YSec

January 2, 2024

Protocol Audit Report

YSec

March 7, 2023

Prepared by: YSec

Lead Auditors:

- Yanagi57

Table of Contents

Protocol Summary

A user should be able to set a password, and retrieve it. Other users should not be able to see my password.

Disclaimer

The YSec 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
Likelihood	High	H	H/M	M
	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:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

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

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 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 below test case shows how anyone can read the password directly from the blockchain.

1. Spin up a new local chain

```
anvil
```

2. On a separate tab

```
make deploy
```

3. 1 represents the second slot of the storage, in this case, the private password field

```
cast storage 0x<address> 1 --rpc-url <anvil-url>
```

The output should be the password in hex.

4. Cast the output into string

```
cast parse-bytes32-string <output-hex>
```

We should get `myPassword`

Recommended Mitigation: Due to this, the overall architecture 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. 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 the password.

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

Description: The `PasswordStore::setPassword` function is set to be an `external` function, howeverm the natspec of the function and overall purpose of the smart contract is that `This function allows only the user to set a new password.`

```
     /*
@>   * @notice This function allows only the owner to set a new password.
      * @param newPassword The new password to set.
      */
      //? can a non-owner set the password?
      //? should a non-owner be able to set a password
      //@audit-high any user can set a password - Missing access control
     function setPassword(string memory newPassword) external {
         s_password = newPassword;
         emit SetNetPassword();
     }
```

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

Proof of Concept:

Code

```
    function test_anyone_can_test_password(address randomAddress)
        public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "newPassword";
        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.
     //@audit there is no newPassword parameter - Suggested action:
        remove the below line
@>   * @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;
    }
```

Impact: the natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3        //@audit there is no newPassword parameter - Suggested action:
             remove the below line
4   -    * @param newPassword The new password to set.
5        */
6       function getPassword() external view returns (string memory) {
7           if (msg.sender != s_owner) {
8               revert PasswordStore__NotOwner();
9           }
10          return s_password;
11      }
```