Protocol Audit Report

Version 1.0

BL

December 14, 2023

Protocol Audit Report

BLaeir

14/12/2023

Prepared by: BLaeir Lead Security Researcher: - BLaeir

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 BLa 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

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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

Add some notes about how the audit went, types of things found and etc.

We spent X hours with Z auditors using Y tools. etc

Issues found

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

Findings

High

[H-1] Password is still on-chain and not acutally private

Description: All data stored on-chain is visible and readable to anyone on the blockchain. The `PasswordStore::s_password` varibale is supposed to be private and only accessed through the `PasswordStore::getPassword` function which is supposed to only be called by the owner of the contract

Proof of visibility on-chain below:

Impact: Anyone can read the private password on the blockhain which breaks the whole 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 make anvil
2. Deploy the contract to the chain make deploy
3. Run the storage tool we use 1 because thats the storage slot where `s_password` is stored in the contract

cast storage 1 –rpc-url http://127.0.0.1:8545

You'll get an output like this: `0x6d7950617373776f726400000000000000000000000000000000000000000014` (roughly)

You can then parse that hex to a string with: cast parse-bytes32-string 0x6d7950617373776f7264000000000000000000000

And get an output of: myPassword

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. You 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 likely also want to remove the view functions as you wouldn't want a user to accidentally send a transaction with the password that decrypts your password.

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

Description: The Password::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 a new password.'

```
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 which severly breaks the contracts intended functionality.

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

```
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
     * @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 signature is `getPassword()` which the natspec say it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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

Gas