Protocol Audit Report

Version 1.0

ReinaBaz

December 27, 2024

Protocol Audit Report

Reina Baz

December 26, 2024

Prepared by: Reina Baz

Table of Contents

Protocol Summary

PasswordStore is a protocol specifically designed for securely storing and retrieving a single user's password. It is intended for use by a single individual, meaning the owner is the sole user of the protocol. Only the owner has the ability to set and access the stored password.

Disclaimer

The Reina 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 describes 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.
- Outsides: No one else should be able to set or read the password.

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, which can be only accessed through the `getPassword` function, which is intended to be only called by the owner of the contract.

We show one method of reading any data off chain below

Impact: Anyone can read the password, severely breaking the functionality of the password.

Proof of Concept:

The below test show how anyone can read the password directly from the blockchain.

1. Create a locally running chain:

```
make anvil
```

2. Deploy the contract on-chain:

```
make deploy
```

3. Run the storage tool using the contract address:

```
cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

we use 1 because that's the storage slot of the `s_password` in the contract.

You'll get an outcome:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. Parse the bytes32 outcome and convert it to string using cast:

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

5. The string outcome is the password which is: `myPassword`

Recommended Mitigation: Due to this, the overall structure of the contract should be rethought. You could encrypt the password offchain, and then store the encrypted password onchain. However, this would require the user to remember another password offchain in order to decrypt this one. You'd also 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 to be an external function, however, based on the natspec and the overall purpose of this smart contract, only the owner should be able 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, severely breaking the contract's 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` 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
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()`, while it should be `getPassword(string)` according to the natspec.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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