Security := Priority

Protocol Audit Report

Version 1.0

Dhanesh Gujrathi

February 26, 2024

Protocol Audit Report

Dhanesh Gujrathi

February 26, 2024

Prepared by: Dhanesh Lead Security Researcher: - Dhanesh

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.

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Disclaimer

The DG Security 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:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Executive Summary

Issues found

Severity	Number of issues Found
High	2
Medium	0
Low	0
Info	1
Gas Optimizations	0
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone & is not actually private

Description: All the data stored on-chain is visible to anyone & can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable & should only be accessed through the `PasswordStore::getPassword` function, which is intended only to be called by the contract owner.

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

Impact: Anyone can read the private password, which severely breaks the functionality of the protocol.

Proof of Concept: (Proof of Code)

The below test case shows how anyone can read the private 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` variable in the contract.

```
cast storage <Deployed Contract Address> 1 --rpc-url http
    ://127.0.0.1:8545
```

You will get an output that looks like -

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Parse the Hex to a String

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

You will get the output -

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be re-thought. One could encrypt the password off-chain & then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the stored password. However you may also 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 check for access control, meaning even a non-owner can change the password

Description: The `PasswordStore::setPassword` function is set as an `external` function with no checks for access control, whereas the natspecs & the whole purpose of this smart contract suggests that `Only the owner should be able to set a new password`.

```
    function setPassword(string memory newPassword) external {
@>      // @audit - There are no access control checks to verify the
   owner
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set / change the password for the contract, which severely breaks the contract's intended functionality.

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

Code

```
    function test_anyone_can_set_password(address randomAddress) public
         {
        vm.assume(randomAddress != owner);

        string memory newPassword = "newPassword";
        vm.prank(randomAddress);
        passwordStore.setPassword(newPassword);

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

Recommended Mitigation: Add an access control condition to the `PasswordStore::setPassword` function.

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, making the natspec incorrect