Password Store Protocol Audit Report

Version 1.0

Demhack

June 26, 2024

Passwrod Store Protocol Audit Report

Demhack

June 26, 2024

Prepared by: Demhack Lead Auditors: - Ahmed Abo-Abdallah (Demhack)

Table of Contents

Protocol Summary

This Password Store Protocol is supposed to be a safe storage for the owner where he can store his passsword and no one except him can retrieve it.

Disclaimer

The Demhack 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 to 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

We spent 1 hour with 1 auditor using manual review and managed to find some critical issues affecting the intended functionality of the contract

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Gas	0
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 fromthe blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accessed through `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

Impact: Anyone can read the private password, severly 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. Create a locally running chain

```
anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Read the storage of the contract

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

You will get an output that looks like this

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Parse the hex to string

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

to get an output of: `myPassword`

Recommended Mitigation: Due to this the overall architecture of the contract should be rethought. 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 password.

[H-2] `PasswordStore::setPassword` has no access control, meaning anyone can change the password

Description: The `PasswordStore::setPassword` function is set to `external` and doesn't have any checks to check if the one calling it is actually the owner

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

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

code

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

```
6       vm.startPrank(owner);
7       string memory actualPassword = passwordStore.getPassword();
8       vm.stopPrank();
9       assertEq(actualPassword, newPassword);
10  }
```

Recommended Mitigation: Add an access control conditional to `setPassword` function.

```
1   if (msg.sender != s_owner) {
2       revert PasswordStore__NotOwner();
3   }
```

Informational

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

Description:

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3        * @param newPassword The new password to set.
4        */
5       function getPassword() external view returns (string memory) {
```

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

Impact: natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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