PasswordStore Audit Report

Version 1.0

Trashpirate.io

November 11, 2024

PasswordStore Audit Report

Trashpirate.io

September 28, 2024

Prepared by: Trashpirate Lead Auditors: - Nadina Oates

Table of Contents

Protocol Summary

A smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The TRASHPIRATE 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 desribed 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

- Based on our thorough review of the code, we have identified a severe conceptual flaw in the protocol and we recommend that the team completely rethinks the architecture of the contract.

We spent 2 hours with 1 auditor using Solidity metrics and static analysis tools to review the code.

Issues found

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

Findings

High

[H-1] Storing password on-chain is public - password is NOT private

Description: All data stored on-chain is visible to everyone and can be read directly from the blockchain. The `Password::s_password` variable is intended to be a private variabled and only accessed through the `Password::getPassword()` function. The `Password::getPassword()` function is intended to be only called by the contract owner.

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. Create a local network:

```
make anvil
```

2. Deploy the contract:

```
make deploy
```

3. Run the storage tool:

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 \
--rpc-url http://127.0.0.1:8545
```

The output:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Convert storage value to string:

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

The output:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One solution might be to encrypto the password before storing it on-chain. But this would require the user to remember another password to encrypto the password that is stored on chain. It should also be throught through for what reason the password should be stored on-chain. If the whole point of the protocol is to possibly protect some data, other methods might be more suitable.

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

Description: The `PasswordStore::setPassword` function is set to be an `external` function, but the natspace of the function and the overall purpose of the function is that `This function allows only the owner to set a new password.`

```
function setPassword(string memory newPassword) external {
    // @autit - There are no access controls!
    s_password = newPassword;
```

```
4       emit SetNewPassword();
5   }
```

Impact: Anone can set/change the password of the contract, severely breaking the functionality of the protocol.

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

Code

```
1  function test_non_owner_can_set_password(address randomUser) public {
2      vm.assume(randomUser != owner);
3
4      vm.prank(randomUser);
5      string memory expectedPassword = "myNewPassword";
6      passwordStore.setPassword(expectedPassword);
7
8      vm.prank(owner);
9      string memory actualPassword = passwordStore.getPassword();
10     assertEq(actualPassword, expectedPassword);
11 }
```

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

```
1  + if (msg.sender != s_owner) revert PasswordStore__NotOwner();
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist - natspec is 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) {
6          if (msg.sender != s_owner) {
7              revert PasswordStore__NotOwner();
8          }
9          return s_password;
10     }
```

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line:

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