Protocol Audit Report

Version 1.0

Rick.io

March 12, 2024

PasswordDEX Protocol Audit Report

Rick

March 12, 2024

Prepared by: Rick Lead Auditors:

- Rick Andronache

Table of Contents

- Informational
  - [I-1] The `Password::getPassword` natspec indicates a parameter that doesn't exist causing the natspec to be incorrect
    * Likelihood & Impact:

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's password. 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

Rick's 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

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 read the password.

Executive Summary

Add some notes about how the audit went, types of things you found 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] Storing the pwd on chain makes it visible to anyone, no matter they keyword visibility

Description: All data store 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 and only be ac-

cessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

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

Proof of Concept: (Proof of Code)

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract on chain

```
make deploy
```

3. Run the local storage tool

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

You will get an output that looks like this

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string value

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

And get the output of `myPassword`

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One 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 would also likely 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.

Likelihood & Impact:

- Impact: HIGH (Worst offenders -> Least bad)
- Likelihood: HIGH
- Severity: HIGH

[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 the overall purpose of the smart contract is that `This function allows only the owner to set a new password`.

```
1      function setPassword(string memory newPassword) external {
2  @>      // @audit - there is no access control here
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: Anyone can set/change the password of the contract, severely breaking the contract intended functionality

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

Code

```
1      function test_anyone_can_set_password(address randomAddress) public
           {
2          // Assume we're not owner
3          vm.assume(randomAddress != owner);
4
5          vm.prank(randomAddress);
6          string memory expectedPassword = "newPassword";
7          passwordStore.setPassword(expectedPassword);
8
9          vm.prank(owner);
10         string memory actualPassword = passwordStore.getPassword();
11
12         assertEq(expectedPassword, actualPassword);
13     }
```

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

Code

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

Likelihood & Impact

- Impact: HIGH
- Likelihood: HIGH

- Severity: HIGH

Informational

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

Description: `Password::getPassword` function signature is `getPassword()` which the natspec say it should be `getPassword(string)`

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3        * @audit there is no newPassword parameter
4    @>  * @param newPassword The new password to set.
5        */
6       function getPassword() external view returns (string memory) {
7           if (msg.sender != s_owner) {
8               revert PasswordStore__NotOwner();
9           }
10          return s_password;
11      }
```

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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

Likelihood & Impact:

- Impact: None
- Likelihood: None
- Severity: Informational / Gas / Non-crits