Protocol Audit Report

Version 1.0

0xNelli

June 6, 2024

PasswordStore Protocol Audit Report

0xNelli

June 2, 2024

Prepared by: 0xNelli

Lead Auditors:

- 0xNelli

Table of Contents

- Informational
  - [I-1] The `PasswordStore::getPassword` function natspec indicates that a parameter should be passed when it is not required and is therefore incorrect

Protocol Summary

PasswordStore is a protocol dedicated to storing and retrieving a user's password off of the blockchain. The protocol is desiged to be used by a single user, only the owner should be able to set and access their password.

Disclaimer

As an independent auditor, I 0xNelli make all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibility for the findings provided in this document. A security audit by myself 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 fundings 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

This security review revealed that the protocol ultimately requires a ground up rearchitecting, as the current implementation does not achieve its goal of securely storing user's passwords. Additionally, the roles stated by the team are not enforced properly, however, this is ulimately made redundant by the failure to secure user's passwords in the first place. Whilst the bockchain facilitates many new and exciting use cases, storage of secure data is not one of them.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Informational	1
Total	3

Findings

High

[H-1] Storage variable data is accessible publically, user's passwords are insecure and viewable

Description: All data stored on-chain is publically visible and can be viewed by anyone. The `PasswordStore::s_password` variable is intended to be private and viewable only by the owner of the contract via the `PasswordStore::GetPassword` function. However, because it is stored on-chain even if the getter function is restricted to be accessible only to the owner. The storage of the contract can be inspected to retrieve the password of the owner.

We will show the process of retrieving this data below.

Impact: Anyone can read passwords in this contract violating the core condition of This violates the condition of 'Others should not be able to access the password.'

Proof of Concept: (Proof of Code)

The below test case list the steps for anyone to view the password variable of the contract:

1. Create a locally running chain

```
make anvil
```

2. Deploy contract

```
make deploy
```

3. Use the cast storage inspection tool

```
cast storage <contract_address> 1 --rpc-url 127.0.0.1:8545
```

4. Convert the `bytes32` value to a readable `string`

```
cast parse-bytes32-string <previous_val>
```

Result will be the password, e.g. using current deploy script:

```
myPassword
```

Recommended Mitigation: This is a critical bug that stems from the architecture of the system. Off-chain methods could be used to supplement the protocol ensuring only encrypted passwords are stored in the contract. However, this requires the user to still need some other key to decrypt the

password stored in the contract. The view function would also have to ensure that the user isn't providing the encryption key to see their password, this must be done off-chain.

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

Description: The `PasswordStore::setPassword` function has visibility of external with no check on the `msg.sender` of the transaction. The natspec of the function and overall purpose of the contract is to ensure that `only the owner can get and set the password stored in this contract`, which this issue violates.

```
function setPassword(string memory newPassword) external {
->  // @audit - Missing access controls
    s_password = newPassword;
    emit SetNetPassword();
}
```

We will show the process of changing the password below.

Impact: Anyone can change the value of `s_password`. Severly breaking the contract intended functionality.

Proof of Concept: (Proof of Code)

The below test case list the steps for anyone to view the password variable of the contract:

1. Create a locally running chain

```
make anvil
```

2. Deploy contract

```
make deploy
```

3. Use the cast storage inspection tool

```
cast storage <contract_address> 1 --rpc-url 127.0.0.1:8545
```

4. Convert the `bytes32` value to a readable `string`

```
cast parse-bytes32-string <previous_val>
```

Result will be the password, e.g. using current deploy script:

```
myPassowrd
```

Recommended Mitigation: This is a critical bug that stems from the architecture of the system. Off-chain methods could be used to supplement the protocol ensuring only encrypted passwords are stored in the contract. However, this requires the user to still need some other key to decrypt the password stored in the contract. The view function would also have to ensure that the user isn't providing the encryption key to see their password, this must be done off-chain.

Informational

[I-1] The `PasswordStore::getPassword` function natspec indicates that a parameter should be passed when it is not required and is therefore incorrect

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

The `PasswordStore::getPassword` signature is `getPassword()` however the natspec indicates that it should be `getPassword(newPassword)`, where `newPassword` is the new password, incorrectly.

Impact: the natspec is incorrect.

Recommended Mitigation:

```
/*
* @notice This allows only the owner to retrieve the password.
-   * @param newPassword The new password to set.
*/
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
```

```
9       return s_password;
10  }
```