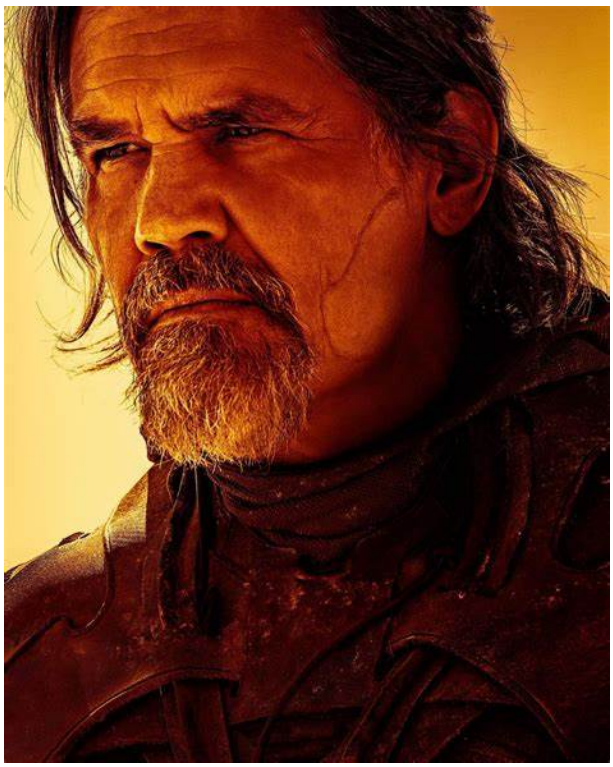

# Protocol Audit Report

Version 1.0

*NerfZeri*

June 3, 2024

# Protocol Audit Report

NerfZeri

june, 6, 2023

Prepared by: [NerfZeri] Lead Auditors: - NerfZeri

## 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 NerfZeri 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 details of this report are all from the following commit hash:

- Commit Hash: 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

### 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

During the Audit Process a total of 3 issues were found and details below.

### Issues found

| Severity | Isuuses Found |
|---|---|
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Info | 1 |
| 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 from the blockchain. the `PasswordStore::s_password` variable is intended to be private information and only accessed through the `PasswordStore::getPassword` function. this is intended to be only accessioble from the owner of the contract.

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

**Impact:** Anyone can read the private password, severley 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 an Anvil chain
2. deploy contract locally
3. run line of code `cast storge <contract Address> 1 --rpc-url http://127.0.0.1:8545`
4. this returns the bytes32 of the variable stored in storage slot 1, being the s_password variable.
5. run line of code 'cast parse-bytes32-string
6. will return the string stored in storage slot 1

**Recommended Mitigation:**

Due to this issue, the architecture of the contract should be rethought as having the sotred password accessible by anyone renders the contract useless for its desired functionallity.

### [H-2] `PasswordStore::setPassword` has no access controls, meaning a non-user could change the password

**Description:** The `PasswordStore::setPassword` function is set to be an `external` function, meaning anyone can call on this function who has access to the contract address. Having no access controls, e.g. `require msg.sender onlyOwner`, leaves this function open so anyone can change the `PasswordStore::s_password` variable.

**Impact:** Anyone can change the `PasswordStore::s_password` variable which is intended to be only accessed by the owner.

```
1      function setPassword(string memory newPassword) external {
2  @>      // missing access control
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

**Proof of Concept:** (Proof of Code) Add the following test to the test suite

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

This will pass showing that any random address can change the password so when the owner calls `PasswordStore::getPassword` it will be equal to the changed password.

**Recommended Mitigation:**

One way the amend this is to add a require statement similar to the one in `PasswordStore::getPassword` this will revert the function if the msg.sender isnt the s_owner.

```
1      function setPassword(string memory newPassword) external {
2          require(msg.sender == s_owner, "PasswordStore__NotOwner");
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

## Informational

### [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesnt exist, causing the natspec to be incorrect.

**Description:** The natspec in the function `PasswordStore::getPassword` indicates a paramater newPassword is to be used yet no paramter is defined in the function.

**Impact:** Potential confusion of someone reading the contract expecting to see a parameter used.

**Proof of Concept:**

```
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 indicated line above shows the included line in the natspec.

**Recommended Mitigation:** Remove the line in the natspec if it the function is not intended to have a `newPassword` paramter, or refactor the function to include the specified paramter.