# Password Store Audit Report

Prepared by: HEXXA Protocol

# Password Store Audit Report

- Title: Password Store Audit Report
- author: Gaurang Bharadava
- date: December 1, 2024

Prepared by: HEXXA Protocol

- Lead Auditors: Gaurang Bharadava

Assisting Auditors:

- None

# Table of contents

▶ Details

See table

# About Gaurang Bharadava

Gaurang Bharadava is experienced smart contract engineer and security researcher. Building HEXXA Protocol, will provide Smart contract development and security audit.

# Disclaimer

The HEXXA Protocol 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 |

# Audit Details

**The findings described in this document correspond the following commit hash:**

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

## Scope

```
./src/
-- PasswordStore.sol
```

# 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.

## Roles

- Owner: The user who can set the password and read the password.
- Outsides: No one else should be able to set or read the password.

# Executive Summary

## Issues found

| Severity | Number of issues found |
|---|---|
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Info | 1 |
| Gas | 0 |
| Total | 0 |

# Findings

## High

### [H-1] Variables stored on-chain are visible to anyone, no matter the solidity visibility keyword meaning the password is not actually private.

**Description:** All data stored on-chain are visible to anyone, and can be read direclty from blockchain. The `PasswordStore::s_password` variable is intented to be a private variable and only accessed by the `PasswordStore::getPassword` function, which is intended to call by only owner of the contract

**Impact:** Anyone can read the private password, severly breaking the functionality of the protocol.

**Proof of Concept:**

the below test case shows that how anyone can read the data directly from the blockchain

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

We use `1` because that's the storage slot of the `s_password` in the contract.

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

You will get output like this.

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can parse that hex to string with:

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

And you can get output like:

```
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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

### [H-2] `PasswordStore::setPassword` has no access control, meaning that non-owner can change the password.

**Description:** `PasswordStore::setPassword` function is set to be `external` function, However , the netspac of the function and overall perpose of smart contract is that `This function allows only the owner to change the password.`

```
    function setPassword(string memory newPassword) external {
@>      //  @audit - there are no access controll.
        s_password = newPassword;
        emit SetNetPassword();
    }
```

**Impact:** Anyone can change/set the password of the contract, severly breaking the contract intended functionality.

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

▶ Details

```
    function test_non_owner_can_change_the_password(address randomAddress) public
{
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = 'myNewPassword';
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

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

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

## Medium

## Low

## Informational

### [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be 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) {} '''

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

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

Informational: Hey, It is not bug, but you should know..

## Gas