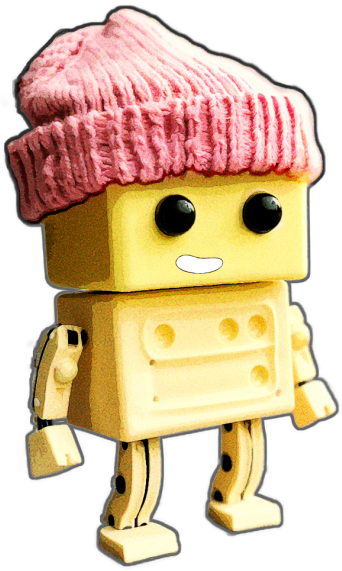

PasswordStore Initial Audit Report

Version 0.1

Cyfrin.io

March 6, 2025

Protocol Audit Report

Akoja

March 6th, 2025

PasswordStore Audit Report

Prepared by: Akoja Lead Auditors: - Abraham akoja

See table

Table of Contents

- Informational
  * [I-1] `PasswordStore::getPassword` natspec indicates a parameter that doesn't exists, causing the natspec to be incorrect

Protocol Summary

A smart contract applicatoin 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

I made all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit by me 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

Commit Hash

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

- Owner: this user can set the password and read the password.
- Outsider: No one else should be able to set or read the password.

Executive Summary

i found 3 vunerabilities, 2 highs and one informational, total time spent was an hour. ## Issues found

Severity	Nunber of issues found
high	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] storing the password onchain makes it visible to anyone.

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 a private variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract. an example of one such method is illustrated below.

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

Proof of Concept: (proof of code)

The below test case proves how any one can read the password directly from the blockchain.

1. Create a local running chain

```
make anvil
```

2. Deploy the contract to the anvil chain

```
make deploy
```

3. Run the storage tool

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

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

you'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

you can the parse that hex to a string with:

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

and get an 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 recall 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 since every data is visible on-chain.

[H-1] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password.

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overall purpose of the smart contract is that `this function allows only owner to set a new password.`

```
function setPassword(string memory newPassword) external {
@>    // @audit - There are  no access modifiers to ensure only owner
    can call this function.
        s_password = newPassword;
```

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

Impact: any one can call this function and therefore break the contract's functionality severly.

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           vm.assume(randomAddress != owner);
3           vm.startPrank(randomAddress);
4           string memory expectedPassword = "mhjhhjhjhyNenwPassword";
5           passwordStore.setPassword(expectedPassword);
6           vm.stopPrank();
7
8           vm.startPrank(owner);
9           string memory actualPassword = passwordStore.getPassword();
10          assertEq(actualPassword, expectedPassword);
11          console.log(actualPassword,expectedPassword);
12          vm.stopPrank();
13      }
```

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

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

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a parameter that doesn't exists, causing the natspec to be incorrect