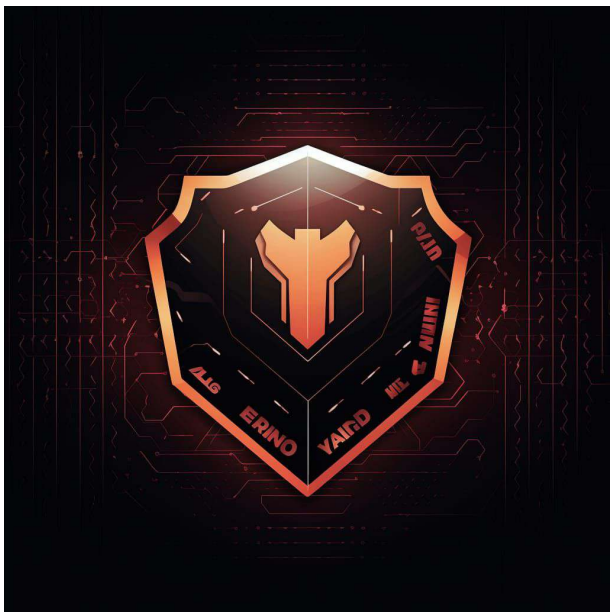

Protocol Audit Report

Version 1.0

Cyfrin.io

December 28, 2023

PasswordStore Protocol Audit Report

Mihai B.

March 7, 2023

Prepared by:Cyfrin Lead Auditors: - Mihai B.

Table of Contents

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

The auditor Mihai B. 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

```
Commit HASH 7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

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
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone, and no longer private

Description: The state variable `PasswordStore::s_password` is intended to be private, meaning it should only be accessible through a specific function (`PasswordStore::getPassword`), and only by the owner of the contract.

However, because all data on the blockchain is public, anyone can read the value of `PasswordStore::s_password` directly from the blockchain, bypassing the `PasswordStore::getPassword` function and the contract's ownership check. This is a significant issue if the contract is intended to store a secret password.

Impact: Anyone can read the private password, severly 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.

We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

Create a locally running chain

make anvil

Deploy the contract to the chain

make deploy

Run the storage tool

We use 1 because that's the storage slot of spassword 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 then parse that hex to a string with:

```
cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000
```

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 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 acces controls, meaning a non-owner can change the password

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

```
1 function setPassword(string memory newPassword) external {
2 @>     // @Audit - There are no Access Controls.
3     s_password = newPassword;
4     emit SetNewPassword();
5 }
```

Impact: Anyone can set/change the stored password, severly breaking the contract's intended functionality

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

Code

```
function test_any_address_can_set_password(address randomAddress)
    public {
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation: Add an access control conditional to the 'setpassword' function.

```
if (msg.sender != s_owner) revert("PasswordStore: Not owner");
```

Medium

Low

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect (Root Cause + Impact)

Description: `passwordStore_getPassword` is the function signature, whereas the Natspec suggests the function should be `getPassword` with a string. The divergence results in incorrect NatSpec.

```
/*
     * @notice This allows only the owner to retrieve the password.
 @>  * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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