Prepared by: Okware Lewis Lead Auditors:

- xxxxxxx

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The YOUR_NAME_HERE 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

	Impact		
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this docment correspond tthe following commit hash

```
7658756936549365
```

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 read or set the password.

Executive Summary

Notes about how the audit went and things we found, etc.

We spent z hours with Y auditors using Y tools.

Issues found

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

Findings

High

[H-1] Storing the password onchain makes it public data, thus accesible by anyone.

Description: All data stored on-chain is visible to anyone and can be read publicly accessed. The variable `PasswordStore::s_password` is intended as a private variable and only accesed through the `PasswordStore::getPassword` function, which is intended to be called only by the owner of the contract.

We demonstrate one such method of reading any data fom the blockchain below.

Impact: Anyone can read the private password, severely 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 on-chain.

1. create local chain by running anvil

```
make anvil
```

2. Deploy contrct to chain

```
make deploy
```

3. Run the storage tool

we use `1` because that is the storage slot for the variable `s_password` in the contract.

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

You will get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with;

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

And you will get an output of :

```
myPassword
```

Recommended Mitigation: Due to this, the overall smart contract architecture should be re-thought. One could encrypt the password off-chain and store the encrypted password on-chain. This would, however, require the user to remember another password off chain to decypt the password. In addition you'd 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.

[H-2] TITLE `PasswordStore::setPassword` function does not have proper access control meaning a non-owner could change the password

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

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

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

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

▶ Code

```
        function test_anyone_can_set_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 an access control conditional to the `PasswordStore::setPassword` function.

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesnt 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()` which the natspec indicates should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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