[H-1] STORING THE PASSWORD ON CHAIN MAKES IT VISIBLE TO EVERYONE, HENCE NO LONGER PRIVATE

Description: All data stored on-chain is viible to anyone, and can be read directly from the blockchain.the s_password vairable is intented to be a private vaariable and only accessed to be a getPassword function, which is internded to be only called by the owner of the contract.

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

Impact: Anyone can read the string(password) hhence it is severly breaking gthe functionality of the protocol

Proof of Concept: (proof of code)

The below test case shows how nyone can read the passwork directly from the blockchain

1. make a running blockchain

```
anvil
```

2. deploy the script

```
make deploy
```

3. run the storage tool

we used `1` because that's the storage slot of `s_passsword` in the contract.

4.

```
cast storage <address_here> 1 --rpc-url http://127.0.0.1.8545
```

5. you will get these output `0x6d7950617373776f726400000000000000000000000000000000000000000014`
6. turn it back to a string using cast `cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000014`
7. run the terminal `myPassword`

Recommended Mitigation: due to this, the overall architural of the contract should be rethought. one could encrpyt thr password off chainn and then store the the on chain data. however you will want to remove the view function as you wouldnt want the user to accidentally send a transaction with the password that decrypts your password.

Likelihood & Impact

Impact: HIGH Likelihood: HIGH Severity: HIGH

[H-2] `password: :setpassword` TITLE SET PASSWRORD HAS NO ACCESS CONTROLS

Description: The `password: :setpassword` function is set to be an external function however, the natspec of the function and overaall purposes of the

start contract is that `this function allow only the owner to set a new password.`

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

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

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

code

```
    ////////////////AUDIT TEST ////////////////////////////////
    ////////////////////////////////////////////////////////////////
    function testAnyoneCanSetPassword(address randomAddress) public  {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

        vm.startPrank(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 (errorPasswordStoreNotOwner);
}
```

Likelihood & Impact

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH ### [SH-N] nespec indicate a parameter that doesnt exists heance causing the nespec 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.sol: :getPassword() nespec indicates the function getPassword() has a parameter, but it doesn't.

Impact: The nespec is incorrect.

Recommended Mitigation: Remove the incorrect nespec line:

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

Likelihood & Impact

- Impact: NONE
- Likelihood: HIGH
- Severity: INFORMATION/GAS/NON-CRITS