[H-1] Storing the password on-chain 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 private variable and only accessed through the `PasswordStore::s_password` function, which is intended 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 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 the blockchain

1. Create a locally running blockchain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool `1` is the storage slot of `s_password` in the contract

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

you will get this output; `0x6d7950617373776f72640000000000000000000000000000000000000000000014`

which you can then parse to a hex string with;

```
cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000000001
```

to get an output of;

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One can encrypt the password offchain, then store the encrypted one on-chain. This would require the user to remember another password off-chain to decrypt the password. However, 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

### Likelihood and Impact: - Impact: HIGH - Likelihood: HIGH - Severity: HIGH

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password

Description: The `Password::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 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 set/change password of the contract , severely breaking the contract's intended functionality

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

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 `setPassword` function

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

Likelihood and Impact

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

[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.
     // @audit there is no newPassword parameter
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()` which the natspec say 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.
```

Likelihood and Impact

- Impact: HIGH
- Likelihood: NONE
- Severity: Informational/Gas/Non-crit

Informational: Hey, this isn't a bug, but you should know…