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

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::get_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, severly breaking the functionality of the protocol.

Proof of Concept:

The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the 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 will look like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

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

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 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.

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

Description: The `PasswordStore::setPassword` function is set to be an external function. However, the natspec of the function and the 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 {
    // There are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

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

Proof of Concept: Add the following to the `PasswordStore.t.sol` 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 condition to the `PasswordStore::setPassword` function.

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

[I-1] The `PasswordStore::getPassword` natspec indicates a aparameter that doesn't 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.
     //@audit no parameter. documentation error
     */
    function getPassword() external view returns (string memory) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

The `PasswordStore::getPassword` function signature is `getPassword()`, while the natspec says 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.
```