[H-1] variables stored in the storage on-chain are visible to anyone, no matter the solidity visibility keyword

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 provate variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract

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

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:
`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

```
cast parse-bytes21-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 create another 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 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 `PasswordStore::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 = new Password;
        emit SetNetPassword();
    }
```

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

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

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

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();
    assetEq(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

[S-#] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.

Description:

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation:

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

Likelihood and Impact:

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