[H-1] Storing 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 `PassworStore::s_password` variable is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be called by the owner of the contract.

We show one such method of reading any off-chain below.

Impact: Anyone cna read the private password, severly breaking the functionality fo 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 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 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
http://127.0.0.1:8545
```

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to string with:

```
cast to-uft8
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 passowrd. 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 the password.

Likelihood & Impact:

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

[H-1] `PasswordStore::setPassword` has not access controls, meaning a non-owner could change the password

Description: `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overall purpose of the contract is that `This function allows only the owner to set a new password`.

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

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

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

▶ Details

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 & Impact:

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

[I-1] The `PasswordStore::getPassword` natspec indivates a parameter 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.
     */
    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.

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

Likelihood & Impact:

- Impact: HIGH
- Likelihood: None
- Severity: Informational/Gas