SECURITY REPORT: PasswordStore	VERSION: v1
DATE: 30-07-2025	AUDITOR: 0xJund

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

Description: All data stored on the blockchain 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::getPassword` function which should only be called by the owner of the contract.

We should one such method of reading any data off the chain below.

Impact: Anyone can read the private password breaking the functionality of storing a "private" password which severly impacts the goal 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 local chain using Anvil

```
make anvil
```

2. Deploy the contract

```
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://127.0.0.1:85
```

Generates the output of:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

Parse the hex into a string using cast:

```
cast parse-bytes32-string 0x6d7950617373776f72640000000000000000000000000000000000000000
```

Which gives an output of:

```
myPassword
```

Recommended Mitigation: Due to the nature of how storage and visibility works the current implementation of the projects make it difficult to continue in the current state.

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

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

```
/* @notice This function allows only the owner to set a new password.
 * @param newPassword The new password to set.
 */
 function setPassword(string memory newPassword) external {
     // @audit there are no access controls
     s_password = newPassword;
     emit SetNetPassword();
 }
```

Impact: Anyone can set/change the password of the contract severly breaking the contracts intended purpose of only allowing the `owner` to set a new password.

Proof of Concept: 1. Adding the following to the foundry test file `PasswordStore.t.sol`

Fuzz test for access control

```
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()
}
```

[I-1] The `PasswordStore::getPassword` natspec refers to a parameter that doesn't exist

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
-->  * @param newPassword The new password to set.
     */
    // @audit there is no newPassword parameter
    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()` which the natspec states it should be ‘getPassword(string)’

Impact: The natspec is incorrect.

Recommended Mitigation:

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