Title: Passwordstore audit file

Author: Koyah Koyahness

[H-1] Storing the password on-chain makes it visible to anyone, compromising privacy

Description: All data stored on-chain is public and visible to anyone, regardless of variable visibility (e.g., `private` or `internal`). The `s_password` variable is intended to be hidden and only accessible by the owner through the `getPassword` function. However, anyone can query the contract's storage slots directly to retrieve the value.

Impact: The intended privacy of the protocol is completely bypassed. Anyone can read the "private" password, rendering the contract's primary purpose ineffective.

Proof of Concept:

Assuming the password `myPassword` is set during deployment, an attacker can use [Foundry's `cast](https://www.google.com/search?q=%5Bhttps://book.getfoundry.sh/cast/%5D(https://book.getfoundry.sh/cast/))` tool to read the storage slot.

1. **Start a local node:**

```
anvil
```

2. **Deploy the contract:**

```
make deploy
```

3. **Read the storage slot** (assuming the password is in slot 1):

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

Output: `0x6d7950617373776f726400000000000000000000000000000000000000000014`
4. **Parse the hex string to plain text:**

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

Output: `myPassword`

Recommended Mitigation: Rethink the protocol architecture. If a password must be stored on-chain, it should be encrypted off-chain first. The user would then store the ciphertext on-chain. Note that the decryption key must never be sent in a transaction or stored on-chain, as it would also be publicly visible.

[H-2] `PasswordStore::setPassword` lacks access control, allowing unauthorized users to change the password

Description: The `PasswordStore::setPassword` function is declared as `external`, but it lacks any access control checks (e.g., an `onlyOwner` modifier). While the function's NatSpec states that "This function allows only the owner to set a new password," the implementation allows any address to call it.

```
function setPassword(string memory newPassword) external {
    // @Audit - No Access Control
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact: Any user can overwrite the stored password, leading to a complete loss of data integrity and breaking the contract's core logic.

Proof of Concept:

Place the following fuzz test in your testing suite to confirm that a non-owner can successfully change the password:

```
function test_anyone_can_set_password(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.startPrank(randomAddress);

    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);
    vm.stopPrank();

    assertEq(passwordStore.getPassword(), expectedPassword);
}
```

Recommended Mitigation: Implement access control using an `onlyOwner` modifier or a manual check within the function:

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

[L-1] Incorrect NatSpec documentation in `PasswordStore::getPassword`

Description: The NatSpec for `PasswordStore::getPassword` includes a `@param` tag for a parameter that does not exist in the function signature.

```
/*
 * @notice This allows only the owner to retrieve the password.
 * @param newPassword The new password to set. <--- This parameter does
     not exist
 */
function getPassword() external view returns (string memory) {}
```

Impact: This results in incorrect documentation, which can mislead developers or automated tools.

Proof of Concept: Not applicable (documentation error).

Recommended Mitigation: Remove the erroneous `@param` line from the NatSpec.

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

Key Changes Made:

- **Severity Labels:** Added `[H-1]`, `[H-2]`, and `[L-1]` (High/Low) to denote the severity of the findings.