# Protocol Audit Report

Version 1.0

*EagleEye*

April 5, 2025

# Protocol Audit Report

EagleEye

April 5, 2025

Prepared by: EagleEye

Lead Auditors: - SteveG

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High

## Protocol Summary

The PasswordStore contract assumes that only the owner can set the password. The `setPassword()` function modifies the `s_password` storage variable, where the password is set, but doesn't include access control meaning that anyone, including a malicious actor, can reset the owner's password.

## Disclaimer

The EagleEye team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

## Risk Classification

| | | Impact | | |
|---|---|---|---|---|
| | | High | Medium | Low |
| | High | H | H/M | M |
| Likelihood | Medium | H/M | M | M/L |
| | Low | M | M/L | L |

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

## Audit Details

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

### Scope

```
src/
--- PasswordStore.sol
```

### Roles

## Executive Summary

### Issues found

| Severity | Number of issues found |
|---|---|
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Info | 0 |
| Gas Optimizations | 0 |
| **Total** | **2** |

## Findings

### High

#### [H-1] Storing the password on-chain with the `private` keyword still makes it visible to anyone

**Description:**
All data stored on-chain can be read. This implies that the supposed `PasswordStore::s_password` variable can be read by anyone, not only through the function `PasswordStore::get_password`.

**Impact:**
The main functionality of the protocol is to store a password that is private to everyone except the owner. However, with this vulnerability, the intention of the protocol is severely breached.

**Proof of Concept:**

```
# 1. Start a local blockchain using Anvil
anvil

# 2. Deploy the contract on the blockchain
make deploy

```

```
# 3. By Solidity convention, `PasswordStore::s_password` is stored in `
    Storage slot, index 1`
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1

# The result is the bytecode representation of the password.
# We can convert it to a string using:
cast --parse-bytes32-string 6
    d79506173737776f72640000000000000000000000000000000000000000000014
```

**Recommended Mitigation:** The main functionality of the protocol is to save passwords on-chain. However, there is no confidentiality on a public blockchain. It is advisable to encrypt the password off-chain and store only the ciphertext on-chain.

Additionally, the key to decrypt this ciphertext must be securely stored off-chain, ensuring it follows all best practices linked here.

### [H-2] Anybody Can Set a New Password, Changing the State of `PasswordStore::s_password`

**Description:**
Without access control on the `PasswordStore::setPassword` function, anybody can set a new password, thereby changing the state of `PasswordStore::s_password`, overwriting the existing password, and disrupting its integrity.

```

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

**Impact:**
- Breach of password integrity.
- Unauthorized modification of stored passwords.

**Proof of Concept:**
Run this test function:

```
function test_anybody_can_set_password() public {
    // Set password as a non-owner.
    vm.startPrank(address(1));
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);
    vm.stopPrank();

    // Retrieve and validate password as owner.
    vm.startPrank(owner);
    string memory password = passwordStore.getPassword();
    assertEq(password, expectedPassword);
}
```

**Recommended Mitigation:** Implement proper access control to ensure only the contract owner can set a new password:

```
function setPassword(string memory newPassword) external {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    s_password = newPassword;
    emit SetNewPassword();
}
```