# Protocol Audit Report

Version 1.0

*0x1057*

February 14, 2025

# Protocol Audit Report

4183

February, 13 2025

Prepared by: 0x1057 Lead Security Researcher: - 0x1057

## Table of Contents

## Protocol Summary

PasswordStore is a smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password

## Disclaimer

0x1057(801) 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 |
| Likelihood | High | H | H/M | M |
| | 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 to the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

### Scope

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

### Roles

- Owner: The user who can set the password and read the password.
- -Outsiders: No one else should be ale to set or read the password.

## Executive Summary

We spent about an hour in manual review to find these issues.

### Issues found

| Severity | Number of Issues Found |
|---|---|
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Info | 1 |
| Total | 3 |

## Findings

### High

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

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

**Impact:** Anyone can read the private password, severely 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 contractt 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 that looks like this: `0x6d7950617373776f726400000000000000000000000000000000000`

You can then parse that hex to a string like so:

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000014
```

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 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 your password.

### [H-2] TITLE `PasswordStore::setPassword` has no access controls, meanng a non-owner could change the password

## Informational

### [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.

**Description:** The `getPassword` function signature is `getPassword()` which the netspec says should be `getPassword(string)`

**Impact:** The natspec is incorrect.

**Recommended Mitigation:** Remove the incorrect natspec line

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