

# Password Store Audit

Kaloyan

August 10, 2025



# PasswordStore Audit

Version 1.0

*Kaloyan*

August 10, 2025

# Password Store Audit

Kaloyan

August 10, 2025

Prepared by: Lead Auditors: - Kaloyan

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    - \* [H-1] Storing the password on-chain makes it visible to anyone, and no longer private
  - Likelihood and Impact: HIGH
    - \* [H-2] `PasswordStore::setPassword` can only be set by owner, but it has no access controls
  - Likelihood and Impact: HIGH
  - Informational
    - \* [I-1] `@param newPassword` doesn't exist but according to the `natSpec` it exists
  - Likelihood and Impact: HIGH

## Protocol Summary

`PasswordStore` is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

## Disclaimer

Kalsito 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		
Likelihood	High	High	Medium	Low
	Medium	H	H/M	M
	Low	H/M	M	M/L
		M	M/L	L

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

## Audit Details

The full audit reports are available here: [View Audit Reports](#)

## Scope

`#-- PasswordStore.sol`

## Roles

Owner: The user who can set the password and read the password.

Outsiders: No one else should be able to set or read the password. `#` Executive Summary

The security review was conducted over a period of approximately 2 hours.

## Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1

Severity	Number of issues found
<b>Total</b>	<b>3</b>

## Findings

### High

#### [H-1] Storing the 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 `PasswordStore::_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:

The following steps demonstrate how anyone can read the `s_password` variable directly from the blockchain storage:

1. Start a local blockchain instance
2. Deploy the contract (Ensure it's deployed to the local node started by Anvil.)
3. Read the storage slot The `s_password` variable is stored in slot 1. Use cast to read it:

```
cast storage 1 --rpc-url http://127.0.0.1:8545 Example output:
```

```
0x6d7970617373776f726400000000000000000000000000000000000000000000
```

4. Parse the storage value into a string

```
cast parse-bytes32-string 0x6d7970617373776f726400000000000000000000000000000000000000000000
```

Output:

```
myPassword
```

This confirms that the private password can be retrieved directly from on-chain storage without calling the contract's getter function.

**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.

### Likelihood and Impact: HIGH

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH ### [H-2] PasswordStore::setPassword can only be set by owner, but it has no access controls

#### Description:

PasswordStore::setPassword is set to be external but in the natSpec it is said to be This function allows only the owner to set a new password.

```
function setPassword(string memory newPassword) external {
    s_password = newPassword;
@>    // @audit - There are no access controls
    emit SetNewPassword();
}
```

**Impact:** Anyone can set/change the password of the contract

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

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 and Impact: HIGH

- Impact: HIGH
- Likelihood: HIGH

- Severity: HIGH ## Informational ### [I-1] @param newPassword doesn't exist but according to the natSpec it exists **Description:**  
 \* @notice This function allows only the owner to set a new password.

@> \* @param newPassword The new password to set.

The PasswordStore::getPassword function signature is getPassword() which the natSpec says it should be getPassword(string).

**Impact:**

The natSpec is incorrect.

**Recommended Mitigation:** Remove the incorrect natSpec line.

+ \* @param newPassword The new password to set.

**Likelihood and Impact: HIGH**

- Impact: HIGH
- Likelihood: NONE
- Severity: Informational/Gas/Non-critical Informational: Hey, this isn't a bug, but you should know...