

# OxJaiveer

## **Protocol Audit Report**

Version 1.0

October 24, 2025

# PasswordStore Audit Report

Jaiveer Singh

October 23, 2025

Prepared by: Jaiveer Singh Lead Security Researcher: - Jaiveer Singh

## 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
    - \* [H-2] No access controls in `PasswordStore::setPassword`, allows anyone to set the password
  - Informational
    - \* [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

## 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

Jaiveer Singh 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

Commit Hash:

```
1 7d55682ddc4301a7b13ae9413095feffd9924566
```

## Scope

```
1 ./src/  
2 #-- 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

I found 2 high severity vulnerabilities in this protocol and 1 informational vulnerability.

## 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 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 : s_password` variable is intended to be a private variable and only accessed through the `PasswordStore : getPassword` function, which is intended to only be 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

```
1 make anvil
```

2. Deploy the contract to the chain

```
1 make deploy
```

3. Run the storage tool

We use 1 because that is the storage location of `PasswordStore::s_password` in the contract

```
1 cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this: `0x6d7950617373776f726400`

You can then parse that hex to string with:

```
1 cast bytes-to-string 0
   x6d7950617373776f72640000000000000000000000000000000000000000000014
```

and get an output of:

```
1 myPassword
```

**Recommended Mitigation:** Because of this issue, the entire architecture of the contract should be rethought. Storing plaintext secrets on a public blockchain is inherently insecure, as all on-chain data is publicly accessible. If the system's purpose requires handling passwords or other sensitive information, this logic must be redesigned to ensure confidentiality. One practical approach is to encrypt or hash the password off-chain using a secure algorithm (e.g., AES encryption or a salted hash via a KDF such as Argon2) and then store only the resulting ciphertext or hash on-chain. The decryption key or preimage should remain entirely off-chain and under the control of the authorized entity. Alternatively, the protocol could employ zero-knowledge proofs to verify that a user knows the correct password without ever revealing it. These design changes maintain functional integrity while ensuring that no private data is publicly exposed.

**[H-2] No access controls in PasswordStore::setPassword, allows anyone to set the password**

**Description:** The `PasswordStore::setPassword` is set to be an `external` function, and the natspec of the function and the overall purpose of the smart contract is that `This allows only the owner to retrieve the password..` However the `PasswordStore::setPassword` lacks any access control mechanism which basically allows anyone to set the password by calling that function.

```
1 function setPassword(string memory newPassword) external {
2     @> // @audit - There are no access controls
3     s_password = newPassword;
4     emit SetNewPassword();
5 }
```

**Impact:** Anyone can set/change the password of the contract by calling the `PasswordStore::setPassword` function, severely breaking the contract's intended functionality.

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

Code

```
1 function test_anyone_can_set_password(address randomAddress) public {
2     vm.assume(randomAddress != owner);
3     vm.prank(randomAddress);
4     string memory newPass = "HackPassword";
5     passwordStore.setPassword(newPass);
6
7     vm.prank(owner);
8     assertEq(passwordStore.getPassword(), newPass);
9 }
```

**Recommended Mitigation:** Add an access control mitigation to the `setPassword` function.

```
1 if(msg.sender != s_owner){
2     revert PasswordStore__NotOwner();
3 }
```

---

**Informational**

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

**Description:**

```
1  /*
2      * @notice This allows only the owner to retrieve the password.
3  @> * @param newPassword The new password to set.
4      */
```

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

**Impact:** The natspec is incorrect.

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

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

---