



## Protocol Audit Report

Prepared by: Have A Shib

# 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 is visible to anyone and no longer private](#)
    - [\[H-2\] PasswordStore::setPassword is not protected by access control; anyone can set the password](#)
  - [Informational](#)
    - [\[I-1\] PasswordStore::getPassword has a misleading @param in the NatSpec comment](#)
- [Gas](#)

# Protocol Summary

---

The `PasswordStore` protocol is a decentralized application (dApp) smart contract designed to let users store and update a private password on-chain. Although it claims to keep the password hidden from others, any data stored on-chain is publicly accessible, making the implementation inherently insecure if passwords are stored in plaintext.

# Disclaimer

---

The Have A Shib 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

Impact				
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 in this document corespond 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 able to set or read the password.

## Executive Summary

We spent 1 hour to audit this contract using Foundry as our tools for making the Proof Of Concept with only 1 auditor doing the audit for this contract

## Issues found

Severity	number of issued found
High	2
Medium	0
Low	0
Informational	1
Total	3

## Findings

## High

[H-1] Storing the password on-chain is visible to anyone and no longer private

**Root Cause:** Storing the password on-chain exposes it publicly

**Impact:** Password is no longer private

**Description:**

All data stored on the blockchain is visible to anyone. Although in the contract only the **Owner** is intended to view the password, **PasswordStore::s\_password** as used in **PasswordStore::setPassword** is not truly private to the owner.

We will demonstrate how a private variable can still be accessed from the blockchain.

**Impact:**

Anyone can read the private password from the blockchain.

**Proof of Concept:**

First, we need a local chain running:

```
make anvil
```

Next, we deploy the protocol. Fortunately, **PasswordStore** has a **make** command set up for us. Note that the deploy script sets the password to **myPassword**. Open a new terminal and run:

```
make deploy
```

Foundry allows us to inspect the storage of a deployed contract using a simple **cast** command. To do this, we need to determine which storage slot the **s\_password** variable uses.

*proof-of-code1*

With that, we can run the following command (replace with your actual address if needed):

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

The output should be similar to:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

This is the **bytes32** form of the data at storage slot 1. Using another Foundry command, we can decode it:

Output:

### Recommended Mitigation:

[H-2] `PasswordStore::setPassword` is not protected by access control; anyone can set the password

**Description:**

**Impact:**

The owner's password can be changed without their permission.

### Proof of Concept:

N/A – function is callable by anyone by design.

### Recommended Mitigation:

Use a library like OpenZeppelin's `Ownable` or `AccessControl` to restrict access to sensitive functions.

► Suggested Access Control

## Informational

[I-1] PasswordStore::getPassword has a misleading @param in the NatSpec comment

**Description:**

In the NatSpec comment of `PasswordStore::getPassword`, there's a `@param newPassword` tag, which is incorrect since the function does not accept any parameters. This creates confusion for developers reading the contract.

```
/*
 * @notice This allows only the owner to retrieve the password.
@> // @audit there is no parameter to set in the function
 * @param newPassword The new password to set.
 */
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    return s_password;
}
```

**Impact:**

While it doesn't affect functionality, it can mislead developers or auditors reading the contract.

**Recommended Mitigation:**

If the function is not intended to take a parameter, the NatSpec should be corrected by removing the incorrect `@param` line.

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

## Gas

---