# Password Store Audit Report

Jesutofunmi Christianah Ajobo

January 2, 2025



# **Password Store Audit Report**

Version 1.0

Lead Auditor:

Jesutofunmi Christianah Ajobo

January 2, 2025

## **Table of Contents**

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Tool Used
- Executive Summary
  - Issues Found
- Findings
  - High Severity
    - \* [H-1] Password is Publicly Accessible
    - \* [H-2] PasswordStore::setPassword has no access control, meaning a non-owner can change the password
  - Informational
    - \* [I-1] TITLE PasswordStore::getPassword natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.
  - Recommendations
- Conclusion

# **Protocol Summary**

PasswordStore is a protocol dedicated to the storage and retrieval of a user's passwords. The protocol is designed for single-user functionality, where only the owner should be able to set and access the password.

## **Disclaimer**

The CodeFortis team makes all efforts to identify vulnerabilities within the given time frame but does not hold responsibility for findings documented in this report. This audit solely focuses on the security aspects of the Solidity implementation.

### **Risk Classification**

Likelihood	High Impact	Medium Impact	Low Impact
High	Н	H/M	M
Medium	H/M	М	M/L
Low	M	M/L	L

The severity matrix follows the CodeHawks methodology.

## **Audit Details**

## Scope

The findings in this document correspond to the commit hash:

2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

- Source Files:
  - src/PasswordStore.sol

### **Roles**

• Owner: The only authorized user to set and access passwords.

### **Tool Used**

Manual inspection of the contract was performed. No automated tools were used in this audit.

# **Executive Summary**

### **Issues Found**

Severity	Number of issues found
High	2
Medium	0

Severity	Number of issues found
Low	1
Informational	1
Gas Optimization	0

# **Findings**

# **High Severity**

### [H-1] Password is Publicly Accessible

**Description:** All data store on-chain is visible to 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 throught the PasswordStore::getPassword function, which is intended to be only called by the owner of the contract.

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

**Impact:** Anyone is able to read the private password, severly breaking the functionality of the protocol.

**Proof of Concept:** (Proof of Code) The below test case shows that anyone can read the password directly from the blockchain 1. Create a locally running chain

make anvil

2. Deploy the contract to the chain.

make deploy

Password-Store Audit Report

5

3. Run the storage tool. I made use of 1 because it is the storage slot of s\_password

in the contract

cast storage <ADDRESS\_HERE> 1 --rpc-url http://127.0.0.1:8545

You will get an output like this

You can the parse the hex into a string with

cast parse-bytes32-string

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 stored password. However, you're also likely want to remove the view

function as you wouldn't want the user to accidentally send a transaction with this

decryption key.

Likelihood & Impact

■ Impact: HIGH

Likelihood: HIGH

Severity: HIGH

# [H-2] PasswordStore::setPassword has no access control, meaning a nonowner can change the password

**Description:** The PasswordStore::setPassword function is set to be an external function, however the purpose of the smart contract and function's natspec indicate that This function allows only the owner to set a new password.

**Impact:** Anyone can set or change the stored password, severly breaking the contract's intended functionality

**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.startPrank(randomAddress);
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);
    vm.startPrank(owner);
    string memory actualPassword = passwordStore.getPassword();
    assertEq(actualPassword, expectedPassword);
}
```

**Recommended Mitigation:** Add an access control conditional to PasswordStore::setPassword.

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

#### Likelihood & Impact

■ Impact: HIGH

• Likelihood: HIGH

Severity: HIGH

#### Informational

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

#### **Description:**

```
/*
    * Onotice This allows only the owner to retrieve the password.

O> * Oparam newPassword The new password to set.
    */
function getPassword() external view returns (string memory) {}
```

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

```
/*
*@notice This allows only the owner to retrieve the password.
- *@param newPassword The new password to set.
*/
```

#### Likelihood & Impact

■ Impact: HIGH

■ Likelihood: NONE

Severity: INFORMATIONAL/NON-CRITICAL

#### Recommendations

• Introduce access control for sensitive functions.

• Avoid storing sensitive data like plaintext passwords directly on-chain.

• Review all NatSpec comments for accuracy.

# **Conclusion**

The report highlights key vulnerabilities in the PasswordStore protocol and provides actionable recommendations to improve security and functionality.