Protocol Audit Report

Version 1.0

John Umoru

August 13, 2025

Protocol Audit Report

John Umoru

August 13, 2025

Prepared by: John Umoru
Lead Auditors: - John Umoru

Table of Contents

Protocol Summary

PasswordStore is a smart contract applicatoin for storing a password. Users should be able to store a password and then retrieve it later. Idealy, thers should not be able to access the password.

Disclaimer

John Umoru 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
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 described in this document corresponds with the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

./src/ #– PasswordStore.sol

Roles

Owner: The person who deployed the contract.
Outsiders: Other blockchain users.

Executive Summary

John Umoru conducted the audit in 1 hour, using manual review and the forge library, in which he found 2 High 1 informational and 1 gas saving issue.

Issues found

Severity	Number Of Issues
High	2
Medium	0
Low	0
Info	1
Gas	0
Total	3

Findings

High

[H-1] Storing the password on chain makes it visible to anyone, thus not private.

Description All data on chain is visible to anyone the storage variable `PasswordStore::spassword` is intended to be private and only viewed by the owner
of the contract using the `PasswordStore::getPassword()` function.

Impact Anyone can read the private password, thus breaking the main functionality of the contract.

Proof of Concept

1. Deploy the contract: `make deploy`

2. retrieve the cleartext password that was deployed with the script from the blockchain by reading the storage slot: `cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1`
3. convert the retrieved bytes to string: `cast parse-bytes32-string 0x50617373776f726431330000`

Recommended Mitigation:
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.

[H-2] Anyone can call the setPassword function, thus allowing any user to set the password.

Description there is no access control set on the `PasswordStore::setPassword()` function. this means any user can call this function to set the password of the contract owner.

Impact Anyone can set the password, thus breaking the integrity component of the password stored.

Proof of Concept
Add the following to the `PasswordStore.t.sol` file

code

```
function testanyonecansetPassword(address anyone) public {
    vm.assume(anyone != owner);

    vm.prank(anyone);
    string memory newpassword = "johnumoruisaking";
    passwordStore.setPassword(newpassword);
    vm.prank(owner);
    string memory expectedpassword = passwordStore.getPassword();
    assertEq(newpassword, expectedpassword);
}
```

OR

1. Deploy the contract: `make deploy`
2. set the password using `cast send 0x5FbDB2315678afecb367f032d93F642f64180aa3 "setPassword(string)""Password13"--rpc-url http://127.0.0.1:8545 --account default`
3. retrieve the cleartext password that was deployed with the script from the blockchain by reading the storage slot:

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1
```

4. convert the retrieved bytes to string to confirm password was stored using:
```
cast parse-bytes32-string <returned bytes>
```

Recommended Mitigation:
Attach an if statement to only allow calls by the owner of the contract as seen below:

code

```
function setPassword(string memory newPassword) external {
    if (msg.sender != sowner) {
        revert PasswordStoreNotOwner();
    }
    spassword = newPassword;
    emit SetNetPassword();
}
```

Informational

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

Description:

```
/*
 * @notice This allows only the owner to retrieve the password.
@> * @param 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

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

Gas

[G-1] `PasswordStore::s_owner` storage variable is not set to immutable

Description:

```
@>     address private s_owner;
       string private s_password;
```

the variable `PasswordStore::s_owner` can be set to an immutable variable which will reduce gas costs.

Impact: It costs more gas to call the storage variable.

Recommended Mitigation: update code

```
-   address private s_owner;
+   address private immutable i_owner;
```