Protocol Audit Report

Version 1.0

Rooyer

June 2, 2025

Protocol Audit Report

Rooyer

Jun 2, 2025

Prepared by: Rooyer Lead Security Researcher: - My self

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of 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

The YOUR_NAME_HERE 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
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

The findings described in this document correspond 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

Issues found

Severity	Number of issues fouend
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password On-Chain are visable to anyone

Description: All data On-Chain is visable to anyone, Even though the variable "`PasswordStore::s_password`" is private, anyone can read its value directly from the contract's storage and only accessible through 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, 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

```
make anvil
```

2. Deploy the contract to the chain

```
 make deploy
```

3. Run the storage tool We use 1 because that's the storage slot of s_password in the contract.

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

   - You'll get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

   You can then parse that hex to a string with:

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

   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.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password

Description: `PasswordStore::setPassword` is an **external** function and should only be called by a specific role, but anyone can. It doesn't validate whether the caller is the owner, so there's no

filter.

```
    function setPassword(string memory newPassword) external {
        // @audit-issue There is no filter if you are the owner
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can **set/change** the password of the contract, severely breaking the contract intended functionality

Proof of Concept: Add the following test of code 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 expectedPass = "myNewPassword";
    passwordStore.setPassword(expectedPass);

    vm.prank(owner);
    string memory actualPass = passwordStore.getPassword();
    assertEq(actualPass, expectedPass);
}
```

Recommended Mitigation: Add an Access Control conditional to the function `PasswordStore::setPassword` so that the verification is performed

```
if(msg.sender != owner){
    revert Not_Owner();
}
```

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.
*/

```

```
6  function getPassword() external view returns (string memory) {}
```

The `PasswordStore::getPassword` function signature is `**getPassword()**` which the natspec say it should be `getPassword(string)`

Impact: The NatSpec is incorrect

Proof of Concept: There is no proof of concept

Recommended Mitigation: Remove the incorrect natspec line

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