PasswordStore Audit Report

Version 1.0

Jovani Tamayo

August 28, 2024

PasswordStore Audit Report

Jovani Tamayo

August 28, 2024

Prepared by: Jovani Tamayo Lead Auditors: - Jovani Tamayo

Table of Contents

Protocol Summary

PasswordStore is a protocol that provides password storage. A user can create a password and store it for no one, but the owner to view. The protocol is designed to be used by a single user. The user can retreive their password they set. Only the owner should be able to retreive and set passwords.

Disclaimer

The Jovani_Tamayo 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
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 correspond to 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 about 1-2 hours conducting an audit for PasswordStore and making all documentation. A couple vulnerabilities were found that were an immediate threat to the protocol.

Issues found

Severity	Number of Issues Found
High	2
Medium	0
Low	0
Informational	1
Total	3

Findings

High

[H-#1] Storing the password on-chain will be visible to anyone, no longer making it private

Description: All data stored on-chain will be visible to anyone. The variable 'PasswordStore::s_password'is intended to be a private variable and only accessible through the 'PasswordStore::getPassword' function, which is intended to be only called by the owner of the contract.

We show one method of showing any data off chain below.

Impact: Anyone can read the private password, severly breaking the functionality of the protocol.

Proof of Concept: (Proof of Code)

The below test 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 is the storage slot of 's_password' in the contract.

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

You'll get an output that looks like this: '0x6d7950617373776f726400000000000000000000000000000000000000000014' You can 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. Once could encrypt a password off-chain, and then store the password on-chain. The user would be required to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the view function to prevent the user from accidentally sending a transaction with the password that decrypts your password.

[H-#2] TITLE 'PasswordStore::setPassword' has no access control, meaning an outsider can set the password

Description: The 'PasswordStore::setPassword' function has no access control and without one anyone who is not the owner will be able to set the password. This contract requires that the password can only be changed or set by the owner.

```
1      function setPassword(string memory newPassword) external {
2  @>          //@Audit Needs access controls
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: Anyone can set the password which goes against the documentation of the contract, severly breaking the contract intended functionality.

Proof of Concept: Add the following 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 expectedPassword = "Mynewpassword";
        passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation: Add an access control conditional to the 'PasswordStore::setPassword' function.

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

Informational

[I-#3] TITLE The 'PasswordStore::getPassword' natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
@>     /* @param newPassword The new password to set.
     */
    //@Audit There is no parameter for newPassword
    function getPassword() external view returns (string memory) {
```

The 'PasswordStore::getPassword' function signature is 'getPassword()' which the natspec said it should be 'getPassword(string)'.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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