PasswordStore Audit Report

Nirban Chakraborty

December 29, 2023

PasswordStore Initial Audit Report

Version 0.1

N Audits

December 30, 2023

PasswordStore Audit Report

Nirban Chakraborty

December 29, 2023

Prepared by: N Audits Lead Auditor: - Nirban Chakraborty

Table of Contents

Protocol Summary

Password is a protocol dedicated to storage and retreival of a 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

- Commit Hash: 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0

Severity	Number of issues found
Low	1
Info	1
Gas Optimizations	0
Total	0

Findings

High

[H-1] Storing the password on-chain makes it visible to everyone

Description- All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

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

Impact- Anyone can read the private password, severely breaking the functionality of the protocols.

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

```
bash
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 will get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation

Because of the above reason, the structure of the protocol needs to be changed by thinking through the solution. 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 password. Also the view function needs to be removed so that the user dosen't accidentally send a transaction with the password that decrypts your password.

Likelihood & Impact:

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

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

Description- The `PasswordStore::setPassword` function is said to be an `external` function, however, the natspec of the functionand overall purpose of the smart contract is that `This function allows only the owner to set a new password.`

```
solidity
    function setPassword(string memory newPassword) external {
@>      // @audit - There are no access controls
        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 to the `PasswordStore.t.sol` test file.

Code

```
solidity
    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 mitigation to the `setPassword` function.

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

Likelihood & Impact:

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

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 natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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