Protocol Audit Report

Version 1.0

Nengak Emmanuel Goltong

May 24, 2024

Protocol Audit Report

Nengak Emmanuel Goltong

May 24, 2024

Prepared by: Spomaria

Lead Security Researcher: - Nengak Emmanuel Goltong

Table of Contents

Protocol Summary

The PasswordStore Protocol enables user to set a password by saving it on the Blockchain and retrieving it later. Other users should not be able to retrieve password stored by another user.

Disclaimer

Nengak Emmanuel Goltong 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 to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: The user who can set the password and later retrieve the password.
- Outsiders: No one else should be able to set or read the password.

Executive Summary

Issues found

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

Findings

High

[H-1] Storing the password on-Chain makes it visible to anyone, and no longer private

Description: all data stored on-chain is visible 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 through the `PasswordStore::getPassword` function, which is intended to only be called by the owner of the contract.

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

Impact: anyone can read the 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 `anvil`
2. Open another bash terminal and deploy the contract using the below code `make deploy`
3. Run the storage tool Read from storage of the deployed contract using the below command. We use `1` because password is stored in storage slot `1`. `cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http://127.0.0.1:8545` This will return the hex value of `0x6d7950617373776f726400000000000000000000`

   Convert the hex to string using the below command `cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000014` which will give the following output `myPassword`

Recommended Mitigation: Due to this, the overall architecture of the contract should be re-thought. One could encript the password off-chain and then store the encripted password on-chain. This will require the user to remember another password off-chain to decript the password. However, you would likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decripts your password.

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

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

```
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 intended functionality of the contract.

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

Code

```
function test_anyone_can_set_password(address randomAddress) public {
    // a random user attempts to set/change password
```

```
3       vm.assume(randomAddress != owner); // ensure a different address
            from owner is setting/changing the password
4       vm.startPrank(randomAddress);
5       string memory expectedPassword = "myNewPassword";
6       passwordStore.setPassword(expectedPassword);
7       vm.stopPrank();
8
9       vm.prank(owner);
10      string memory actualPassword = passwordStore.getPassword();
11
12      assertEq(expectedPassword, actualPassword);
13  }
```

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

```
1  if(msg.sender != s_owner){
2      revert PasswordStore__NotOwner();
3  }
```

Informational

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

Description:

```
1  /*
2       * @notice This allows only the owner to retrieve the password.
3       * @param newPassword The new password to set.
4       */
5      // @audit function has no parameter - @param is documentation error
6      function getPassword() external view returns (string memory) {}
```

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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