Protocol Audit Report

Naveen Prakash

December 26, 2023

Naveen Audits

Protocol Audit Report

Version 1.0

prakashnaveen473@gmail.com

December 28, 2023

Protocol Audit Report

Naveen Prakash

December 26, 2023

Prepared by: Naveen Prakash Lead Auditors: - Naveen Prakash

Table of Contents

Protocol Summary

This protocol stores password for a user(supposedly also the owner), so that only the user can retrieve it.

Disclaimer

The Naveen Prakash 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

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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

This project was worked on by Naveen Prakash for 14 hours in a time-boxed manner and the finding are presented below

Issues found

```
| Severity | Number of issues found |
| -------- | ---------------------- |
| high     | 2                      |
```

medium	0
low	0
Info	1

Findings

High

[H-1] Any storege variable stored in blockchain is readable irrespective of its accessibility which means that any password stored is also visible even if its 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 only to 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 private password, severly 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
```

1. 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
```

Youll get an output that looks like this:

0x6d79506173737776f72640000000000000000000000000000000000000000014

You can then parse that hex to a string with:

```
cast parse-bytes32-string 0x6d79506173737776f7264000000000000000000000000000000000
```

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 password. However, you vwould also likely want to remove the view function as you wouldnt want the user to accidently send a transaction with the password that decrypts the password.

[H-2] `PasswordStore::setPassword` has no access control due to which anyone(non-owner) can change the password

Description: (Often time it is better to put in the exact code right in the writeup and at time is also good to add something from the docs itself to illustrate your point) 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, severly breaking the contract intended functionality.

Proof of Concept:

Add the following 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 setPasswword function.

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that dosent exist, causing the natspec to be incorrect

```
    /*
     * @notice This allows only the owner to retrieve the password.
=>   * @param newPassword The new password to set.
    */
    function getPassword() external view returns (string memory) {
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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