Protocol Audit Report

Version 1.0

Vkgoud

January 29, 2025

Protocol Audit Report

Vk goud

Jan 29, 2025

Prepared by: Vkgoud

Lead Auditors:

- Vamshi Krishna Goud

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicted to storage and retrieval of a user's password. 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 Vkgoud 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 the follwoing commit hash:

```

7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```

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

Roles

- owner: The user who can set the password and read the password
- Outsides: No one else should be able to set or read the password.

Executive Summary

Issues found

Severiety	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] The Variable stored in storage is on chain and visible to anyone, no matter what the solidity visibility keyword meaning, the password is not actually a private password.

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 private variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

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

//1. Create a locally running chain

`run anvil chain`

//2. make deploy `forge deploy`

// 3. Run the storage tool

we use 1 because that is 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 like this 0x6d7950617373776f726400000000000000000000000000000000000000000014

then parse that hex to a string with

```
cast parse-bytes32-string <Hash>
```

and get the output as string

Proof of Concept: (Proof of code)

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'd also likely want to remove the view function as you wouldn't want the user to accidently send a transcation 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 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 funciton allows only the 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 ca set/change the password of the contract, severly breaking the contract intended functionality.

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

```

        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 `setPassword` functin.

```


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

Likelihood & Impact:

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

Informational

[I-1]`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.
    // @audit there is no nwePassword parameter!

     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

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

Impact: The natspec is incorrect.

Proof of Concept: No POC

Recommended Mitigation: Remove the incorrect natspec line

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

Likelihood & Impact

- Impact : HIGH
- Likelihood : NONE
- Severity : Informational/Gas/non-critic Informational : Hey, this is not a bug but you should know…