Protocol Security Report

Prepared by: Kweks

Prepared by: Kweks

Lead Security Researcher: Kweks

Table of Contents

Protocol Summary

PasswordStore is a protocol that is dedicated to storage and retriveal 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 Kweks 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

	Impact		
	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

The audit was relatively straightforward. I found an access control bug

I spent an hour using tools like foundry, solidity metrics

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Informational	
Total	3

Findings

High

1. [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` varibale 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, 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`
2. Deploy the contract to the chain `make deploy`
3. Run the `storage` tool
4. 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`
5. You'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000000000000014`
6. You can then parse that hex to a string with: `cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000014`
7. And get an output of: `myPassword`

Recommended Mitigation: The overall architecture of the contract should be rethought. The password could be encrypted off-chain, and then store the encrypted password on-chain. The view function should be be removed so that the user does not accidentally send a transaction with the password that decrypts your password.

[H-2] `PaaswordStore::setPassword` has no access controls, meaning a non-owner could 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 cintract is that `This function allows only the owner to set a new password.`

▶ Code

```
        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 functionality.

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

▶ Details

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 `setPassword` function.

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

Informational

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

Description:

▶ Code

```
        /*
     * @notice This allows only the owner to retrieve the password.
@>   * @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 `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

▶ Code

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