Protocol Audit Report

Version 1.0

Alvaro Manganello

January 2, 2024

Protocol Audit Report

Alvaro Manganello

January, 2024

Prepared by: Dapp Guardians Lead Security Researcher: - Alvaro Manganello

Table of Contents

Protocol Summary

A smart contract applicatoin for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The Dapp Guardians 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 following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

- In Scope:

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

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum ## 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 audit focused on assessing the security and functionality of Password Store. The findings highlight critical vulnerabilities that could compromise the confidentiality and integrity of user data.

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 in storage on-chain is visible to anyone, no matter the solidity visibility keyword. Meaning the password can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be private and only access through `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:

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

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: `0x6d7950617373776f726400000000000000000000000000000000`

4. Parse the hex obtained from the blockchain into string:

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

Get an Output: `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 remember another password off-chain to decrypt the first password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no access control, 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 contract is that `This function 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 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 `PasswordStore::setPassword` function.

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

Informational

[I-1] NatSpec indicates that `PasswordStore::getPassword` needs 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 natscep for the function `PasswordStore:getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual signature is `getPassword()`.

Impact: The natsPec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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