PasswordStore Audit Report

Pedro Machado

November 24, 2024

Protocol Audit Report

Version 1.0

Pedro Machado

November 24, 2024

PasswordStore Audit Report

Pedro Machado

November 24, 2024

Prepared by: Pedro Machado Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

This protocol aims to provide a secure way to store and retrieve passwords. It was designed to be used by a single user. Only the owner should be available to set and access this password.

Disclaimer

Pedro Machado as the security researcher of this protocol 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

```
./src/
|___PasswwordStore.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

We spent 8 hours with just one auditor using Fundry framework and making a manual auditing review.

Issues found

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

Findings

High

[H-1] Exposure of Sensitive Data: Misleading Use of private Visibility for Password Storage

Description: All data stored on-chain is visible and readable for everyone. The `'PasswordStore::s_password'` variable is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function, wich is intended to be only called by owner of the contract. Whe 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 case, show how anyone can can read the passoword directly from the blockchain.

1. Create a locally running chain

   ```
   make anvil
   ```

2. Deploy the smart contract `bash make deploy` copy in the clipboard the smart contract address deployed. `0x5FbDB2315678afecb367f032d93F642f64180aa3`

3. Get the data stored.

   Look like we use the `1` argument, beacuse `PasswordStore::s_password` correspond to the storage slot #1 Geneal Use Case.

   ```
   cast storage <ADDRESS_SMART_CONTRACT>  SLOT --rpc-url NETWORK
   ```

   In this case:

   ```
   cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url 127.0.0.1:8545
   ```

   ```
   0x6d7950617373776f726400000000000000000000000000000000000000000014
   ```

4. Parse the retrieved data from bytes32 to string

   ```
   cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000000
   ```

5. Reading from the cli

`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 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 decrypt your password.

[H-2] `PasswordStore::setPassword` has not access controls, meaning a non-owner could change the password

Description: Everyone can update the password through the `PasswordStore::setPassword` function. Despite the natspec actually specify that: `@notice This function allows only the owner to set a new password.` The function does not has access controls mechanism. Resulting a non-owner caller could update the password just calling the function.

```
    function setPassword(string memory newPassword) external {
@>        //@finding there're not access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyyone 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 randomAdress) public {
        vm.assume(randomAdress != owner);
        vm.prank(randomAdress);
        string memory newPassword = "hello_ethereum";
        passwordStore.setPassword(newPassword);
        vm.prank(owner);
        string memory actuallyPasswordd = passwordStore.getPassword();
        assertEq(actuallyPasswordd, newPassword);
    }
```

Recommended Mitigation: Add an access control mechanism to the setPassword function.

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

Informational

[I-1] TITLE The `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.
@>     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

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

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

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