Protocol Audit Report

Version 1.0

Carl.io

July 26, 2025

Protocol Audit Report

Carl

July 25,2025

Prepared by:Carl Lead Security Researcher: - Carl

Table of Contents

Protocol Summary

用于存储密码的智能合约应用程序。用户应该能够存储密码，然后在以后检索它。其他人应该无法访问密码。

Disclaimer

The YOUR_NAME_HERE 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
Likelihood	High	H	H/M	M
	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

Commit Hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

- In Scope:

```
./src/
└── PasswordStore.sol
```

Roles

-Owner:The user who can set password and read the password.

-Outsiders:No one else should be able to set or read the password.

Executive Summary

Issues found

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

Findings

High

[H-1] 密码链上存储导致信息公开，不再私密

Description: 所有链上存储的数据都是公开可见的。`PasswordStore::s_password` 变量旨在隐藏，并仅通过 `PasswordStore::getPassword` 函数供所有者访问。

Impact: 任何人都能够读取私密密码，严重破坏了协议的功能。

Proof of Concept 1. 启动本地区块链：首先，我们需要一个运行中的本地区块链。在终端中执行以下命令启动 `anvil`：

```
Bash

```

```
```
anvil
```

（请注意：大多数概念验证通常不需要本地区块链。）
```

2. **部署协议：**接下来，我们需要部署 `PasswordStore` 协议。幸运的是，该协议提供了 `make deploy` 命令。请在一个新的终端中运行以下命令，该部署脚本会将密码设置为 `myPassword`：

   Bash

   ```
   make deploy
   ```

3. **检查合约存储：** Foundry 允许我们使用 `cast` 命令检查已部署合约的存储。为此，我们需要记住 `s_password` 变量被分配到的存储槽位。在 `PasswordStore` 合约中，`s_password` 通常位于**存储槽位 1**。

使用以下 `cast` 命令读取该存储槽位的数据（请注意，您的合约地址可能会有所不同）：

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
http://127.0.0.1:8545
// 合约部署后给出地址 url是在执行anvil命令后在所在终端内给出的反馈
```

您将收到类似以下的十六进制输出：`0x6d7950617373776f726400000000000000000000000000000000000000000000`

4. **解码数据：**上述输出是存储槽位 1 中数据的字节形式。我们可以使用 Foundry 提供的另一个便捷命令来解码这些数据，将其转换回可读的字符串：

```
Bash

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

此时，您将看到解码后的输出：`myPassword`
```

至此，我们通过几个简单的命令就清楚地展示了客户端期望在链上保持隐藏的数据（即密码）实际上是任何人都可以访问的。这些步骤有力地证明了漏洞的存在。

Recommended Mitigation **考虑到链上数据的公开性质，协议的整体架构需要重新审视。**旨在存储私密信息的协议不应将明文数据直接放在链上。

一种可行的替代方案是：

- **链下加密与链上存储加密数据：**用户可以在链下对密码进行加密，然后将**加密后的密码**存储在链上。
- **链下密钥管理：**这将要求用户妥善保管一个**链下密钥**（即另一个密码），用于解密存储在链上的加密密码。
- **移除不必要的视图函数：**同时，为了避免用户意外地在交易中暴露解密密钥，强烈建议移除 `getPassword` 这样的 `view` 函数，或者将其修改为不直接返回敏感信息。

[H-2] `PasswordStore::setPassword` 没有访问控制，意味着非所有者可以更改密码

Description: `PasswordStore::setPassword` 函数被设置为 `external` 函数，但智能合约的目的和函数的 **natspec** 指示“此函数只允许所有者设置新密码。”

```
function setPassword(string memory newPassword) external {
    // @Audit - There are no Access Controls.
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact: 任何人都可以设置/更改存储的密码，严重破坏了合约的预期功能。

Proof of Concept: 为了证明这个漏洞的存在并确认非所有者能够调用 `setPassword` 函数，我们编写了一个模糊测试 **(fuzz test)**。此测试函数将随机地址作为输入，模拟这些非所有者地址调用 `setPassword` 函数，并验证密码是否被成功更改。该模糊测试在 **256** 次运行中都通过了，这明确表明任何地址都能够成功调用 `setPassword` 函数，从而证实了此访问控制漏洞的存在。

```
function test_anyone_can_set_password(address randomAddress) public {
    // 确保 randomAddress 不是合约所有者
    vm.assume(randomAddress != owner);
    // 模拟 randomAddress 调用 setPassword
    vm.startPrank(randomAddress);
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);

    // 切换回所有者账户，验证密码是否已被更改
    vm.startPrank(owner);
    string memory actualPassword = passwordStore.getPassword();
    // 断言实际密码与预期（被非所有者设置的）密码相同
    assertEq(actualPassword, expectedPassword);
}
```

Recommended Mitigation: 为了解决此漏洞，应向 `setPassword` 函数添加适当的访问控制。最直接的解决方案是使用 **OpenZeppelin** 的 `Ownable` 合约提供的 `onlyOwner` 修饰符。这将确保只有合约的所有者才能成功调用此函数。

```
import "@openzeppelin/contracts/access/Ownable.sol";

contract PasswordStore is Ownable {
    string private s_password;

    constructor(string memory initialPassword) Ownable(msg.sender) {
        s_password = initialPassword;
    }

    // @notice This function allows only the owner to set a new
        password.
    // @param newPassword The new password to set.
    function setPassword(string memory newPassword) external onlyOwner
```

```
            { // 添加 onlyOwner 修饰符
13          s_password = newPassword;
14          emit SetNewPassword();
15      }
16
17      function getPassword() public view returns (string memory) {
18          return s_password;
19      }
20
21      event SetNewPassword();
22  }
```

Informational

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

Description:

```
1  /*
2   * @notice This allows only the owner to retrieve the password.
3   * @param newPassword The new password to set.
4   */
5  function getPassword() external view returns (string memory) {}
```

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`. **Impact:** The natspec is incorrect **Recommended Mitigation:** Remove the incorrect natspec line.

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3  -     * @param newPassword The new password to set.
4        */
```

Gas