Protocol Audit Report

Version 1.0

Cyfrin.io

February 27, 2024

Protocol Audit Report

Ardeshir Gholami

27 feb 2024

Prepared by: Ardeshir Lead Auditors: - Patrick Collins, - Ardeshir :)

Table of Contents

Protocol Summary

PasswordStore is a test protocole which saves a password and lets the only owner read it anytime. # 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

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

`src/ PasswordStore.sol`

Roles

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

Executive Summary

Issues found

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

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone

Description: All data stored chain is visible to anyone and can be read directly from the blockchain. the `PasswordStore::s_password` var is intended to be private and only visible 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:

Proof of Concept: (Proof of Code) 1. Create a local chain

```
make anvil
```

2. Deploy the contract

```
make deploy
```

3. run the storage tool to read storage slots the password is set in storage slot 1

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

4. parse it to string from bytes32

```
ast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

Recommended Mitigation: In the audit of the smart contract, a critical security concern regarding the handling of sensitive information was identified, specifically the recording of a password in a private string. This practice poses a significant risk, as it exposes the system to potential unauthorized access and data breaches. Instead of storing passwords directly in the smart contract, it is recommended to implement a more secure and robust authentication mechanism.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a nonowner can change the password

Description: The `PasswordStore::setPassword` function within the smart contract is designed to allow users to update their stored password. However, upon review, it was discovered that this function lacks any form of access control. This means that any user, not just the owner of the password, can call this function and change the password stored in the contract. This lack of access control poses a significant security risk, as it allows unauthorized users to potentially gain access to sensitive information or manipulate the system in ways that could compromise its integrity and security.

```
    function setPassword(string memory newPassword) external {
        //n why anyone can set a new password ?
        // @audit Any user can set a new password
        // missing access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: The absence of access controls on the PasswordStore::setPassword function could lead to several adverse outcomes. Firstly, it could enable unauthorized users to change passwords without the consent of the original owner, potentially leading to unauthorized access to accounts or sensitive information. Secondly, it could be exploited by malicious actors to disrupt the normal operation of the smart contract or to gain unauthorized access to critical functions or data. This could have far-reaching

implications for the security and functionality of the smart contract, affecting not only the immediate operation but also the trust and reliability of the system in the long term.

Proof of Concept: To demonstrate a proof of concept for the vulnerability identified in the PasswordStore::setPassword function using the provided steps, we'll follow a series of commands to interact with a local Ethereum blockchain, deploy a smart contract, and then attempt to change the password from a non-owner's address. This example assumes you have a basic understanding of Ethereum, smart contracts, and the tools mentioned. add this test to your projects test file:

```
    function testSetPasswordNotOwner(address randomAddress) public {
        // Attempt to set the password as a non-owner
        string memory expectedPassword = "myNewPassword";
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: To address this vulnerability, it is recommended to implement robust access control mechanisms for the `PasswordStore::setPassword` function. This could involve requiring that only the owner of the password can call this function, or by implementing additional authentication and authorization checks to ensure that only authorized users can change the password. Additionally, consider implementing logging and monitoring to detect and alert on unauthorized access attempts, allowing for quicker response and mitigation of potential security incidents. Regular security audits and penetration testing should also be conducted to identify and address any other potential vulnerabilities in the smart contract.

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

Informational

[I-1] `PasswordStore::getPassword` Function Misleading Comments

Description: The `PasswordStore::getPassword` function within the smart contract is intended to allow only the owner to retrieve the stored password. However, the function does not accept any parameters, which contradicts the comment that mentions a parameter for the new password. This discrepancy between the function's implementation and its documentation could lead to confusion

and potentially expose the password to unauthorized access if developers mistakenly assume the function requires a parameter.

Impact: The natspec is incorrect.

Proof of Concept: Recommended Mitigation: Remove the incorrect natspec line,

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