Protocol Audit Report

Version 1.0

Amir Gubaidullin

February 17, 2024

PasswordStore Protocol Audit Report

Amir Gubaidullin

Feb 17, 2024

Prepared by: Amir Gubaidullin Lead Auditors: - Amir Gubaidullin

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single owner and nobody else. Owner must be able to store, change and get password in any time.

Disclaimer

The Amir Gubaidullin 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

Scope

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

Roles

- Owner: The user that can set, change and get password.
- Outsides: No one else should be able to set, change or get password.

Executive Summary

I spend approximately 3 hours do the audit, including the watching of the Patrick Collin's tutorial. I found 2 high severity issues.

Issues found

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

Findings

High

[H-1] Private storage variables are visible on-chain, so the password is not truly private.

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PaswordStore::s_password` variable is intended to be private variable and only accessed from the `PaswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

I will show 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 that 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

```
cast storage <ADDRESS_OF_THE_CONTRACT> 1 --rpc-url http
    ://127.0.0.1:8545
```

After that you can parse hex value you got to the string with:

```
cast parse-bytes32-string <HEX_VALUE>
```

And get the private password :)

Recommended Mitigation: To address the issue of private storage variables being visible on-chain and ensure the confidentiality of sensitive data such as passwords, if is recommended to encrypt sensitive data before storing it on-chain. This ensures that even if the data is visible, it remains unreadable without the decryption key. However, it requires password owner to store another decrypt-key off-chain.

[H-2] Access control in `PasswordStore::setPassword` is missed, meaning a non-owner could change the password

Description: In `PasswordStore::setPassword` function access control is missed. This completely destroys the concept of this smart contract: `This function allows only the owner to set a new password`

```
function setPassword(string memory newPassword) external {
 @>       // @audit Here is NO acces control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set up his own password. Thus, the contract owner will lose his own password that was once set.

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

Code

```
function test_non_owner_can_set_password(address randomAddress) public
    {
        vm.assume(owner != randomAddress);
        vm.startPrank(randomAddress);
        string memory expectedPassword = "nonOwnerPassword";
        passwordStore.setPassword(expectedPassword);

        vm.startPrank(owner);
        string memory actualPassword = passwordStore.getPassword();

        assertEq(actualPassword, expectedPassword);
    }
```

And run the following command to be sure the vulnerability exists:

```
forge test --mt test_non_owner_can_set_password
```

Recommended Mitigation: To mitigate the risk of unauthorized users changing the password, implement access control within the setPassword function to ensure that only the contract owner can modify the password. This can be achieved by adding a modifier or conditional statement that checks the sender's address against the owner's address before allowing the password to be updated. Additionally, ensure thorough testing to verify that access control mechanisms are functioning correctly.

Exaple of modification:

Code

```
function setPassword(string memory newPassword) external {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Medium

Low

Informational

Gas