Protocol Audit Report

Version 1.0

Cyfrin.io

February 21, 2024

PasswordStore Audit Report

AqFatma

February 21, 2024

Prepared by: Fatma Lead Security Researcher: - Fatma

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a users password.The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The AQ 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 to the following commit hash: Commit Hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

Roles

- Owner: the one who can set and read password.
- Outsiders: No one else should be able to read or set the password. # Executive Summary

Spent 1 hour researching vulnerabilities and doing tests using foundry and anvil. ## Issues found

severity	Number Of Issues Found
Highs	2
Mediums	0
Lows	0
Info	1
Totals	3

Findings

High

[H-1] The password is not private to users/hackers. Variable stored in storage on the chain is visible to anyone, no matter the solidity visibility keyword.

Description:

All data stored on the chain is visible to anyone and can be read directly from the blockchain. The `PasswordStore::s_password` variable isn't intended to be private and only accessed through the `PasswordStore::getPassword` function, which is intended to be called only by the owner of the contract.

We show an instance of how to read data off-chain below.

Impact:

This allows anyone to read the password, thus breaching the main functionalities of the protocol.

Proof of Concept: (Proof Of Code)

The below test case shows how one 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

Use 1 because that's the storage slot for `s_password` in the contract.

```
cast storage <Contract Address Here> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this `0x6d7950617373776f726400000000000000000000000000000000000000000`

You can then pass the hex to a string with

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

And get an output of

```
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 the chain. This would require the user to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want to accidentally send a transaction with the password that decrypts password.

[H-2] Missing access control on `PasswordStore::setPassword`, anyone can change the password

Description:

The `PasswordStore::setPassword` function is external and has no access control restriction/-functionality. However, according to the natspec, it describes the smart contract function as `This function allows only the owner to set the password`.

```
function setPassword(string memory newPassword) external {
 //@audit there are no access controls
    s_password = newPassword;
    emit SetNetPassword();
}
```

Impact:

This allows anyone to set or change the password, severely breaking the contract's intended functionality.

Proof of Concept:

Add the following to the `PasswordStore.t.sol` 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 on the `PasswordStore::setPassword` function

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

Informational

[S-#] The `PasswordStore::getPassword` natspec indicates a parameter `newPassword` that does not exist, causing the natspec to be incorrect.

Description:

The `PasswordStore::getPassword` function signature is `getPassword` while the natspec describes it as `getPassword(string)`.

Impact:

The natspec is incorrect.

Recommended Mitigation:

Remove the incorrect natspec line.

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