Dezly Macauley - Security Audit

Protocol Name: PasswordStore
Audit Date: 01 September 2024

Audit Scope:

Contract URL

https://github.com/Cyfrin/3-passwordstore-audit

Commit Hash

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Files in scope

```
./src/PasswordStore.sol
```

Protocol Overview:

---

1. PasswordStore.sol

Purpose

A smart contract application for storing a password.
Users should be able to store a password and then retrieve it later.
Others should not be able to access the password.

Potential Attack Vectors

- Can the user password be prevented from being retrieved later?
- Can I prevent the user's password from being stored?
- Can an unauthorized user access the password?
- Is the password encrypted on both ends, and in transit?

Issue 1 - A Password variable is exposed

- File: `PasswordStore.sol`
- Contract: `PasswordStore`

Likelihood and Impact:

- Impact: High
- Likelihood: High
- Severity: High

Location in code, and high-level overview:

```
// SECTION: State Variables
address private s_owner;

// AUDIT: Exposed password:
// s_password is NOT private on the blockchain
// The 'private' keyword only means that this variable can't be accessed
// by other contracts outside of this file.
string private s_password;
```

Impact:

All data stored on the blockchain is visible and can be read directly by anyone. This compromises the privacy of user passwords and effectively nullifies the purpose of the protocol.

Proof of Concept:

```
// Start anvil local chain in a new terminal
anvil

// Compile and deploy the contract
forge compile
make deploy

// Get the address of the deployed contract:
0x5FbDB2315678afecb367f032d93F642f64180aa3

// To get the value of `s_password` (storage-slot-number 1) in hexadecimal format:
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http://127.0.0.1:8545

// Output (hex format):
0x6d79506173737 76f726400000000000000000000000000000000000000000000000014

// Convert to human-readable string:
cast parse-bytes32-string 0x6d79506173737 76f726400000000000000000000000000000000000000000000000014

// Output:
myPassword
```

Recommended Mitigation:

Encrypt the password off chain and store the encrypted password on chain. Handle the actual decryption and verification of passwords off-chain.

Issue 2 - Missing Access Control

- File: `PasswordStore.sol`
- Contract: `PasswordStore`

Likelihood and Impact:

- Impact: High
- Likelihood: High
- Severity: High

Location in code, and high-level overview:

```
// AUDIT: Attack Vector Found: Missing Access Control
function setPassword(string memory newPassword) external {
    s_password = newPassword;
    emit SetNetPassword();
}
```

Impact:

An unauthorized person could change the password and lock out a valid user, nullifying the purpose of the protocol.

Proof of Concept:

```
function test_unauthorized_password_change(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.prank(randomAddress);
    passwordStore.setPassword("unauthorized_password");
    vm.prank(owner);
    assertEq(passwordStore.getPassword(), "unauthorized_password");
}

forge test --match-test test_unauthorized_password_change
```

Recommended Mitigation:

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

Issue 3 - Documentation Issue (Referring to a non-existent parameter)

- File: `PasswordStore.sol`
- Contract: `PasswordStore`

Likelihood and Impact:

- Impact: None
- Likelihood: Not Applicable
- Severity: Informational

Location in code, and high-level overview:

```
/*
 * @param newPassword The new password to set.
 */
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    return s_password;
}
```

Recommended Mitigation:

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