Password Store Report

Performed by: *Shiki.eth*

March 18, 2024

Table of Contents

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Introduction

A time-boxed security review of the **Password Store** protocol was done by **name**, with a focus on the security aspects of the application's smart contracts implementation.

Protocol Summary

Password Store is a protocol for storing and retrieving a user's password. Only the owner should be able to set and access this password.

Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Audit Details

The findings described in this document corespond to the following commit hash:

```
2e8f81e263b3a9d18fab4fbc46805ffc10a9990
```

Scope

```
- contracts/
  - PasswordStore.sol
```

Roles

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

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password on-chain is visible to anyone and no longer private

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

Impact: Anyone can read the private password, severly breaking the functionality of the protocol.

Proof of Concept: The below test case shows how 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 We use 1 because that's the storage slot of `s_password` in the contract.

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

You will get this output: `0x6d795061737377f7264`

4. Parse the hex output to a string with this command

```
cast parse-bytes32 0
    x6d7950617373776f7264000000000000000000000000000000000000000000000014
```

You will get this output:

```
myPassword
```

Recommended Mitigation: Due to this the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then the encrypted password on-chain. This would require the user to remember another off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner can set the password

Description: The `PasswordStore::setPassword` function is set to be an external function, however, the natspec of the function and overall purpose of the contract is that `This function allows only the owner to set a new password`.

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

Impact: Anyone can set/change the password of the contract, severly breaking the contract intended functionality.

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

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

    vm.stopPrank();

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

Recommended Mitigation: Add an access control conditional to the `setPassword` function

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

Informational

[I-01] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist, causing the natspec to be incorrect

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line

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