Title Page

Protocol Audit Report

Version 1.0

Author: Abdikadr Jerow

Date: July 7, 2024

Protocol Summary

PasswordStore is a protocol dedicated to the storage and retrieval of a user's passwords. The protocol is designed to be used by a single user and is not intended for multiple users. Only the owner should be able to access the passwords stored in the protocol.

Disclaimer

The YOUR_NAME_HERE team makes every effort to find as many vulnerabilities in the code as possible in the given time period but holds no responsibility 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

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

`2e8f81e263b3a9d18fab4fb5c46805ffc10a9990`

Scope

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

Roles

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

Executive Summary

I spent two weeks auditing this contract using various tools like Slither, Mythril, and manual code review. I found that the contract is vulnerable to a reentrancy attack. The attacker can call the `setPassword` function multiple times to drain the contract of its funds. I recommend that the contract owner fix this vulnerability by using the `reentrancyGuard` modifier to prevent reentrancy attacks.

Findings

Issues Found

Severity	Number of Issues Found
High	2
Medium	0
Low	0
Info	1
Total	3

High

[H-1] Storing the password on-chain makes it 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 private and only accessed through the `PasswordStore::getPassword` function, which should be called only by the contract owner.

Impact: Anyone can read the private password, severely compromising the protocol's functionality.

Proof of Concept:
The test case below demonstrates how anyone can read the password from the blockchain.

Create a locally running chain:
```
make anvil
```
1.

Deploy the contract to the chain:
```
make deploy
```
2.

Run the storage tool:
Use `1` because that is the storage slot of `s_password` in the contract.
```
cast storage &lt;ADDRESS_HERE&gt; 1 --rpc-url http://127.0.0.1:8545
```
You will get an output like this:
```
"0x6d7950617373776f726400000000000000000000000000000000000000000014"
```
You can then parse that hex to a string with:
```
cast parse-bytes32-string
0x6d7950617373776f726400000000000000000000000000000000000000000014
```
And get an output of:
```
mypassword
```
3.

Recommended Mitigation: Store the password securely, such as using a key management service or a secure storage solution.

Likelihood & Impact

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

[H-2] The `PasswordStore::setPassword` function has no access controls, allowing non-owners to change the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function. However, the natspec and overall purpose of the smart contract indicate that this function should only allow the owner to set the password.

Impact: Anyone can set/change the password, leading to unauthorized access to the smart contract and its associated data.

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.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 to the `setPassword` function to ensure that only the owner can set the password.

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

Likelihood & Impact

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

Informational

[I-1] The PasswordStore::getPassword natspec indicates a parameter that doesn't 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()`, but the natspec incorrectly indicates it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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