Prepared by: Sakithiya Lead Auditors :

- xxxxxxx

Table of Contents

Protocol Summary

PasswordStore is aprotocol dedicated to storage and retrieval of a user's passwords/ The protocol is designed to be used by a single user. and is not designed to be used by multiple users. Oly the owwner should be able to set and access this password.

Disclaimer

The YOUR_NAME_HERE 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

Likelihood vs Impact	High	Medium	Low
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 the following commit hash:

```

```

Scope

```
src
#-- Paasswordstore.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.

Executive Summary

Add some notes about how the audit went, types of things you fiynd and so.

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 makesit visible to anyone and no longer private

Description: All data stored on-chain is visble to anyone and cn be read directly from the blockchain. The `PasswordStore:: s_password` variable is intended to be a private variable and only access through the `PasswordStore:: getPassword` function which is intended to be 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 steps show how anyone can read the password directly from the blockchain.

- create locally running chain `make anvil`
- Deploy the contract on the chain. `make deploy`
- Run the storage tool. We use `1` because thats the storage slot of `s_password` in the contract.
- cast storage addresshere 1 --rpc-url https://127.0.0.1:8545
- you will get output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000000000000014`
- you can then parse that hex in a string with

```
cast arse-bytes32-string
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

- you will get an output of above command `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-chain. This would require the user to remember another password off-chain to decrypt the stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

Likelihood & Impact Summary

Impact: High **Likelihood:** High **Severity:** High

[H-2] `PasswordStore:setPassword` has no access control, even non-owner could 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 smart contract is that `This function allows only the owner to set a new password.`

```
function setPassword(string memory newPassword) external {
s_password = newPassword;
```

```
emit SetNewPassword();
}
```

Impact: Anyone can set/change the password of the contract that severly breaking the contract intended functionality. **Proof of Concept:** Add the following to the `PasswordStore.t.sol` test 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 to the setPassword function.

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

Likelihood & Impact Summary

Impact: High **Likelihood:** High **Severity:** High

Medium

Low

Informational

[I-1] `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.
```

```
*/
@audit no newpassword parameter set // document issue
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 natpsec is incorrect

Recommended Mitigation:

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

Likelihood & Impact Summary

Impact: None **Likelihood:** High **Severity:** Informational

Gas