Prepared by: Olusola Jaiyeola Lead Auditors:

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrival of user's passwords. the protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

Disclaimer

The Solacodes 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

		Impact		
		High	Medium	Low
Likelihood	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 with the followong commit hash:

```
248e94def42bac6d560cd374569cab58bd0682db
```

Scope

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

Roles

- Owner: The user who can set the password and read the passwrod.
- 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 found, etc

We spent X hours with Z auditors using Y tools. etc

Issues found

Severity	Bumber of issues found
High	2
Medium	0
Low	0

Severity	Bumber of issues found
Info	1
Total	3

Findings

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 variable nd only accessed through the `PasswordStore::getPassword` function, which is intended to be called by the owner of the contract.

We show one such method of reading any data off chain below.

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

Proof of Concept/Code: The below test case shows how anyone can read the password directly from the blockchaain.

1. Create a locally running chain

```
make anvil
```

2. Deploy The contract to the chain

```
  make deploy
```

3. Run 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'll get an output that looks like this:

   `0x6d7950617373776f726400000000000000000000000000000000000000000014`

   You can then parse that hex to a string with:

```
cast parse-bytes32-string
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and store the encrypted password on-chain. This would require the user to remmeber another password 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 control, meaning a non-owner could change the password

Description: The `PasswordStore::setPassword` function is set to be `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 {
@>      //@audit - There are no access controls
          s_password = newPassword;
          emit SetNetPassword();
      }
```

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

Proof of Concept: Add the below 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
    }
```

Informational

[I-1] Rectifying the Natspec: A Revelation in the `PasswordStore::getPassword`

Description:

```
    /*
     * @notice Only the chosen owner may invoke this sacred rite to unveil
the password.
@>    /@audit Alas! There is no trace of the fabled "newPassword"
parameter!
     */
    function getPassword() external view returns (string memory)
```

In the chronicles of PasswordStore::getPassword, a discrepancy arises: while the function bears the seal of getPassword(), the ancient scripture of natspec proclaims it should be `getPassword(string)`.

Impact: A veil of falsehood obscures the true nature of the natspec.

Recommended Correction: Purge the erroneous natspec line from the annals.

```
- * @param newPassword The elusive key to unlock a new password.
```