Prepared by: yavor.eth Lead Security Researcher:

- yavor.eth

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    - [H-1] Storing the password on-chain makes it visible to anyone, and no longer private
    - [H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password.
  - Informational
    - [I-1] TITLE The `PasswordStore::getPassword` natspec insudo apt install texlivedicates a parameter that does not exist, causing the natspec to be incorrect.

Protocol Summary

PasswordStore is a protocol 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. Only the owner should be able to set and access this password.

Disclaimer

The yavor.eth 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
	High	H	H/M	M

		Impact		
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 the following comit hash:

```
Commit hash: 7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

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

*We spent overall about 3 hours, 3 vulnerabilities/bugs were found, we did testing and overlooking the code to make sure everything is accurate.

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

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

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

Proof of Concept: (Proof of Code)

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'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 then store the encrypted password on-chain. This would require the user to remember 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 controls, meaning a non-owner could change the password.

Description: The `PasswordStore::setPassword` which is set to be an external ruins the purpose of the contract. Here only the owner `should set a new password`!

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

Impact: Anyone can input a password which severly breaks the functionality of the contract.

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.startPrank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);


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

Recommended Mitigation: Control the access of `setPassword` function.

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

Informational

[I-1] TITLE The `PasswordStore::getPassword` natspec insudo apt install texlivedicates a parameter that does not exist, causing the natspec to be incorrect.

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
     // @audit their is no newPassword parameter!
     * @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.

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