Password Store Audit

Prepared by: Leonard Trevor

Table of Contents

Protocol Summary

PasswordStore is a smart contract applicatoin for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The diexlabs 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
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

Commit Hash

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- owner
- outsiders

Executive Summary

Issues found

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

Findings

High

[H-1] Variables stored in storage on-chain is publicly visibly to anyone.

Description: All data stored stored onchain 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 be accessible to the owner through the `PasswordStore::getPassword` function.

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

Proof of Concept: (Proof of code)

The below test case shows that anyone can read the password from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

1. 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 the 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 like 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 it onchain but this would require the user to remember another password that they would use to encrypt/decrypt the password off-chain. The view function should

also be removed so as to prevent the user from accidentally sending a transaction containing the secret password.

[H-2] `PasswordStore::setPassword` has no access controls which means a non-owner can change 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 change the password`.

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

Impact: Anyone can change the password stored in the contract which severely brakes the contract.

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

▶ Code

```
    function test_anyone_can_change_the_password(address randomAddress)
public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = 'newPassword';
        passwordStore.setPassword(expectedPassword);

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

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

```
    function getPassword() external view returns (string memory) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist causing the natspec to

be incorrect.

Description:

```
/*
     * @notice This function allows only the owner to set a new password.
@>     * @param newPassword The new password to set.
     */
    function setPassword(string memory newPassword) external
```

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

```
    /*
     * @notice This function allows only the owner to set a new password.
-    * @param newPassword The new password to set.
     */
    function setPassword(string memory newPassword) external
```