Protocol Audit Report

Prepared by: Brittany Winters

Table of Contents

Protocol Summary

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

Brittany 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 me 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

	Impact		
Low	M	M/L	L

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

🚨 Issue Report 1:

Passwords stored on chain are not private.

⚠ Severity

High - 1

📝 Description

All data stored on chain is visible to anyone and can be read 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 only be called by the owner of the contract.

💥 Impact

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

🔬 Proof of Concept

The following test case shows how anyone can read the `PasswordStore::s_password` variable:

We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

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 `PasswordStore::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:

`0x6d7950617373776f7264000000000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

`cast parse-bytes32-string 0x6d7950617373776f7264000000000000000000000000000000000000000000000014`

And get an output of:

`myPassword`

Recommended Mitigation

The current contract architecture needs to be reconsidered. A potential solution could involve encrypting the password off-chain before storing the encrypted version on-chain. This approach, however, would necessitate the user to remember an additional off-chain password for decryption purposes. Additionally, it may be prudent to eliminate the view function to prevent the user from inadvertently transmitting a transaction containing the decryption password.

Issue Report 2:

`PasswordStore::setPassword` is callable by anyone.

Severity

High - 2

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 {
@>      // @audit: There are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact

Anyone can set/change the password of the contract.

Proof of Concept

Add the following function to `PasswordStore.t.sol`

▶ Code

```
        function test_anyone_can_set_password(address randomAddress) public {
        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 modifier to the `setPassword` function.

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