Protocol Audit Report

Prepared by: 0xshuayb

Table of Contents

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 0xshuayb 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

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

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

Findings

[H-1] Storing the password on-chain makes it visible to anyone

Description: All data stored on-chain are visible to anyone and can be read on the blockchain regardless of any solidity keyword. The `PasswordStore::s_password` variable is intended to be a private variable onle to be called by the owner of the contract using the `PasswordStore::getPassword()` function.

Impact: Anyone can read the password stored, defeating the whole functionality of the contract.

Proof of Concept: The test case below shows how anyone can read the stored password on-chain

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

```
cast storage <CONTACT ADDRESS> 1 --rpc-url http://127.0.0.1:8545
```

`1` was used above because that is the storage slot of `s_password` You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`, which is a bytes32 code. It can be parsed to a string using:

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

This gives an output of:

```
myPassword
```

Recommended Mitigation: All data on the blockchain is public. To store sensitive information, additional encryption or off-chain solutions should be considered. Sensitive and personal data should never be stored on the blockchain in plaintext or weakly encrypted or encoded format.

[H-1] `PasswordStore::setPassword()` is missing access control, which implies anyone could change the password

Description: The `PasswordStore::setPassword()` function is set to be an `external` function but the functionality of the contract expects that only the owner can set a new password.

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

Impact: Anyone can change/set password,severely breaking the functionality of the contract

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

▶ Code

```
function TestAnyoneCanSetPassword(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 condition to the `setPassword()` function.

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

[I-1] `PasswordStore::getPassword()` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

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