Protocol Audit Report

Version 1.0

Cyfrin.io

October 22, 2024

Protocol Audit Report

Zhou Hang

October 21, 2024

Prepared by: Zhou Hang Lead Security Researcher: - Zhou Hang

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrival 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 YOUR_NAME_HERE 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:

Commit Hash:

```
?????????
```

Scope

```
- PasswordStore.sol
```

Roles

- Owner: The user who can set the password and read the password.
- Outsides: 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 aduditors using Y tools. etc

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 visable 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 chian below.

Impact: Anyone can read the private password, serverly 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 depoly
```

3. Run the storage tool

We use1 because that's the storage slot of `s_password` in the contract.

```
cast storage <ADDRESS_HERE> 1 --rpc-url http://localhost: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 0
   x6d7950617373776f726400000000000000000000000000000000000000000014
```

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 control, meaning a non-owner could 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 set a new password.`

```
1     function setPassword(string memory newPassword) external {
2 @>      // @audit - There are no access controls
3         s_password = newPassword;
4         emit SetNetPassword();
5     }
```

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

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

/**notice:details summary at least with “‘javascript keep a skip of line */

Code

```
1   function test_anyone_can_set_password(address randomAddress) public {
2         vm.assume(randomAddress != owner);
3         vm.prank(randomAddress);
4         string memory expectedPassword = "myNewPassword";
5         passwordStore.setPassword(expectedPassword);
6         vm.prank(owner);
7         string memory actualPassword = passwordStore.getPassword();
8         assert(keccak256(abi.encodePacked(actualPassword)) == keccak256
              (abi.encodePacked(expectedPassword)));
9     }
```

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

```
1 if(msg.sender != s_owner){
2     revert PasswordStore__NotOwner();
3 }
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that dosen’t exist, causing the natspec to be incorrect

Description:

```
1  /*
2       * @notice This allows only the owner to retrieve the password.
3  @>     * @param newPassword The new password to set.
4       */
5      function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()` which the natspec say it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: remove the incorrect natspec line

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