Protocol Audit Report

Version 1.0

Elyas

July 2, 2024

Protocol Audit Report

Elyas

March 7, 2023

Prepared by: Elyas Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

The `PasswordStore` contract assumes that only the owner can set the password. The `setPassword()` function modifies the `s_password` storage variable, where the password is set, but doesn't include access control meaning that anyone, including a malicious actor, can reset the owner's 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
Likelihood	High	H	H/M	M
	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 finding described in this document corrospond the following commit hash:

commit hash

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
#-- 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

Issues found

Severity	Number of issu 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 chain below.

Impact: Any one can read the private password, severly breaking functionality of the protocol

Proof of Concept: (proof of code)

The below test case shows how anyone can read the password directly form 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 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 controls, meaning a non-owner could change the password

Description: The `PasswordStore::setPassword` function is set to be `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 thers is no access controll
3         s_password = newPassword;
4         emit SetNetPassword();
5     }
```

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

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

Code

```
1     function test_anyone_can_set_password(address randomAddress) public
          {
2         vm.assume(randomAddress != owner);
3         vm.prank(randomAddress);
4         string memory expectPassword = "myNewPassword";
5         passwordStore.setPassword(expectPassword);
6 
7         vm.prank(owner);
8         string memory actualPassword = passwordStore.getPassword();
9         assertEq(expectPassword, actualPassword);
10    }
```

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()` indicates a parameter that does not exist, causing natspec to be incorrect.