Protocol Audit Report

Version 1.0

Martin K.

September 2, 2024

Store Password Audit Report

Martin Kolev

September 2, 2024

Prepared by: [Martin Kolev] Lead Security Researcher: - Martin Kolev

Table of Contents

Protocol Summary

Protocol does X, Y, Z

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

Scope

Roles

Executive Summary

Issues found

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` is intented to be a private variable and only accessed through `PasswordStore::getPasswordPassword` function, which is intented to be only called by the owner of the contract.

Impact: Anyone can read the private password, severly breaking the functionallity of the protocol.

Proof of Concept: (Proof of Code) The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

Create a locally running chain `make anvil` Deploy the contract to the chain `make deploy` 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 0x6d7950617373776f726400000000000000000000000000000000000000000`
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] Fuction for updating the password can be called by anyone, meaning non-owner can change the password

Description: The `PasswordStore::setPassword` function has not the needed access control, but it is intented to be called only by the owner of the contract.

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

Impact: Non-owner can change the password calling `PasswordStore::setPassword` function

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 expectedPassword = "myNewPassword";
5          passwordStore.setPassword(expectedPassword);
6      }
```

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

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

Medium

Low

Informational

Gas