Protocol Audit Report

Version 1.0

Pedro PINTO

May 15, 2024

Protocol Audit Report

Pedro PINTO

May 15, 2024

Prepared by: Pedro

Lead Auditors: - Pedro

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 Pedro 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:

```
53ca9cb1808e58d3f14d5853aada6364177f6e53
```

Scope

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

Roles

- Owner: The user who can set the password and read the password.
- Outsiders: Not one else should be able to set or read the password.
-

Executive Summary

I'm spent 4 hours using Foundry tool.

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 chain below.

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

Proof of Concept: The below test case shows how anyone can read the password directly from the blockchain.

1. Create a local chain with anvil: `bash window anvil or make anvil`
2. Deploy the contract to chain local chain with a new bash window: `new bash window make deploy or @forge script script/DeployPasswordStore.s.sol: DeployPasswordStore $(NETWORK_ARGS)`
3. Run the storage tool with cast: `` `1` `` `refers to the position of `s_password` in the storage slot of the contract.` bash window cast storage CONTRACT `ADDRESS DEPLOYED LOCALY` 1 –rpc-url `ADDRESS OF THE LOCAL ANVIL CHAIN` ‘`We get an output like this one:` 0x6d7950617373776f726400000000000000000000000000000000000000000014‘
4. Find the string: To get the password string, we need to parse it from bytes32 to string like this: `cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000` Finally, we will get from this bytes32 output this string: `myPassword`

Recommended Mitigation: Due to this, the overall architecture of this contract must be rethought. One could directly to encrypt the password on-chain with a Fully Homomorphic Encryption from the open-source Zama’s library.

[H-2] `PasswordStore::setPassword`has no access control, meaning a non-owner could change the password

Informational

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

Gas