Protocol Audit Report

Version 1.0

Shurjeel

January 23, 2024

Protocol Audit Report

Shurjeel Khan

Jan 20, 2024

Prepared by: Shurjeel Lead Auditors: - Shurrjeel Khan

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 Shurjeel 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

High

[H-1] Storing the password on-chain makes it visable to anyone, and no longer private

Description: All the data is visable to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only access through the `PasswordStore::getPassword` function, which is intended to be called by only 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.

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 becasue that's storage slot of `s_password`

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

You will get the output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

Then you can parse that hex to string with:

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

And get output of:

```
myPassword
```

Proof of Concept:

The test case below shows how anyone can read the passaword directly from the blockchain.

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-2] Has no acces controls, meaning non-owner can change the password

Description: The `PasswordStore::setPassword` set to be the `external` function, however the natspec of the fuction and overall purpose of the smart contract `This function allow only the owner to set password`

```

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

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

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

Code

```
1
2  function test_anybody_can_call_setPassword(address randomAddress)
       public {
3      vm.assume(randomAddress != owner);
4      vm.prank(randomAddress);
5
6      string memory expectedPossword = "newPassword";
7      passwordStore.setPassword(expectedPossword);
8
9      vm.prank(owner);
10     string memory actualPassword = passwordStore.getPassword();
11     assertEq(actualPassword, expectedPossword);
12 }
```

Recommended Mitigation: Add access control condition to the `setPassword` function

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

Informational

[I-3] Natspec says The `PasswordStore::getPassword()` indicates a parameter that doesn't exits, causing the Natspec to be incorrect.

Description:

```
1
2  /*
3       * @notice This allows only the owner to retrieve the password.
4  @>     * @param newPassword The new password to set.
5       */
6      function getPassword() external view returns (string memory) {
7          if (msg.sender != s_owner) {
8              revert PasswordStore__NotOwner();
9          }
10         return s_password;
11     }
```

The `PasswordStore::getPassword` function signature is `getPassword()` while the Natspec says it should be `getPassword(string)`

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec

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