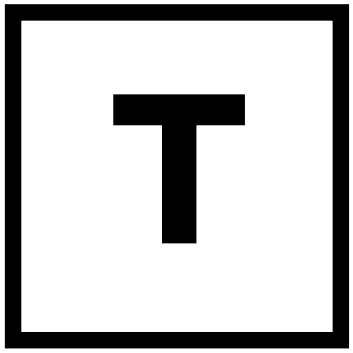

Protocol Audit Report

Version 1.0

Tamayo

October 13, 2024

Protocol Audit Report

Tamayo

March 7, 2023

Prepared by: [Tamayo] Lead Auditors: - Jovani Tamayo

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 Tamayo 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 to the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
PasswordStore.sol
```

Roles

- Owner: The user who can set the password and read the password.
- Outsiders: No one else should be able to set or read the password.

Executive Summary

2 Highs and 1 Informational found

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 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 and data off chain below

Impact: Anyone can read the private password, severely 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 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 storgage <contract address> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like:

'0x6d7950617373776f726400000000000000000000000000000000000000000014'

4. Parse the hexadecimal into string

```
cast parse-bytes32-string <hexadecimal representation>
```

And get an output of:

'myPassword'

Recommended Mitigation:

Do not use private to make a variable private to the eyes of others since as shown above is not actually private on-chain. You should rethink the structure of your codebase and refactor it to make passwords hidden another way such as utilizing encryption off-chain.

[H-2] There is no access control for 'PasswordStore::setPassword' meaning anyone can set a password

Description: In 'PasswordStore::setPassword' there is no modifier such as onlyOwner restricting other users from setting a password. This is a violation to the protocols documentation stating only users can set and access a password.

```
    function setPassword(string memory newPassword) external {
@>      // @audit Missing access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone or 'outsiders' can set a password contrary to the owners password.

Proof of Concept: Add the following to the 'PasswordStore.t.sol' test file.

Code

```
    function test_non_owner_can_set_password(address randomAddress)
        public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "non-owner";
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

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

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

Informational

[I-1] Natspec is incorrect in 'PasswordStore::getPassword' function which indicated a parameter that doesn't exist

Description:

```
      * @notice This allows only the owner to retrieve the password.
     // @audit There is no newPassword in parameters
@>    * @param newPassword The new password to set.
      */
    function getPassword() external view returns (string memory) {
```

The 'PasswordStore::getPassword' function needs to contain a string parameter within 'getPassword()' according to the natspec.

Impact: Natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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