Protocol Audit Report

Version 1.0

github.com/MaverickStoic

March 12, 2024

Protocol Audit Report

Arash Amiri

March 12, 2024

Prepared by: Arash Lead Auditors: - xxxxxxx

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
  - Findings
  - High
    * [H-1] Variables stored on chain are visible to anyone, no matter the visibility keyword, so the private password is not actually private;
  - Likelihood & Impact:
    * [H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password
  - Likelihood & Impact:
  - Informational
    * [I-1] The `PasswordStore::setpassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.
  - Likelihood & Impact:

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's password. The protocol is designed to be used by a single user, not multiple users. So, only the owner should be able to access and set the password.

Disclaimer

The Arash Amiri 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: Commit Hash:

```
```

Scope

```
./src/
___PasswordStore.sol
```

Roles

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

Executive Summary

How the audit went. Spent x hours, number of auditors…using Y tools. We found some bugs …. ## Issues found | Severity | Number of issues found | | ——– | ———————- | | HIGH | 2 | | MEDIUM | 0 | | LOW | 0 | | INFO | 1 | | TOTAL | 3 | ## Findings ## High ### [H-1] Variables stored on chain are visible to anyone, no matter the visibility keyword, so the private password is not actually private;

Description All data stored on-chain is visible to anyone, and can be read from the blockchain. The `PasswordStore::s_password` variable is intended to be private, and only accessed through the `PasswordStore::setPassword()` function, which is only supposed to be called by the owner of the contract. We show one such method of reading data below.

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

Proof of Concepts

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

1. Create a locally running chain.

```
anvil
```

2. Deploy the contract on 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> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this: `0x6d7950617373776f72640000000000000000000000000000000`

You can then parse the 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. 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.

Likelihood & Impact:

- Impact : HIGH
- Likelihood : HIGH
- Severity : HIGH

[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 an external function; however, the natspec of the function and the overall purpose of the smart contract is that `This function allows only the owner to set the password.`

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

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

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

Code

```
    function testAnyoneCanSetPassword(address randomAddress) public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        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();
}
```

Likelihood & Impact:

- Impact : HIGH
- Likelihood : HIGH
- Severity : HIGH

Informational

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

Description

```
    // @audit There is no getPassword parameter.
    function getPassword() external view returns (string memory) {
    }
```

The `PasswordStore::setpassword` function signature is `getPassword` while the natspec says should be `getPassword(string)`. **Impact** The natspec is incorrect.

Recommended mitigation Remove the incorrect natspec line.

```
-     * @param newPassword the new password to set
```

Likelihood & Impact:

- Impact : NONE
- Likelihood : HIGH
- Severity : Informational/Gas/Non-Crits