Protocol Audit Report

Version 1.0

Dec3mber

August 29, 2024

Protocol Audit Report

Dec3mber

August 28, 2024

Prepared by: Dec3mber

Lead Security Reseacher: Dec3mber

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 Dec3mber 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:

```
7d55682ddc4301a7b13ae9413095feffd992456
```

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

we spend 1 hour using manual review, and successfully identified three vulnerabilities that pose potential risks to the security and functionality of the system.

Each of these vulnerabilities has been categorized based on its severity and potential impact on the overall security of the smart contract. Our detailed analysis and recommended mitigation strategies are provided in the following sections of this report. Addressing these issues promptly will be crucial to ensuring the safety and reliability of the smart contract.

Issues found

Severity	Numbers of issues found
High	2
Medium	0
Low	0
Informational	1
gas	0
total	3

Findings

High

[H-1] Storing the password on-chain makes it visable to anyone, and no more 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 though the `PasswordStore::getPassword` function, which is intended to be noly called by the owner of the contrac.

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: (Proof of Code)

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 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 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. 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] `PasswordStore::setPassword` has no access controls, means a non-owner could change the password

Description: The `PasswordStore::setPassword` function is set to be an external function, without a access control, so everyone can call the function and change the password, fail to meet `This function allows only the owner to set a new password`.

```
1      function setPassword(string memory newPassword) external {
2  @>      // @audit there is no access control
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

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

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

Code

```
1      function test_anyone_can_set_any_password(address randomAddress,
           string memory randomPassword) public {
2          vm.assume(randomAddress != owner);
3          string memory ownerPassword = "myPassword";
4          vm.prank(randomAddress);
5          passwordStore.setPassword(randomPassword);
6
7          vm.prank(owner);
8          string memory actualPassword = passwordStore.getPassword();
9          assert(keccak256(abi.encodePacked(ownerPassword)) != keccak256(
               abi.encodePacked(actualPassword)));
10     }
11
12     function test_anyone_can_set_password(address randomAddress) public
           {
13         vm.assume(randomAddress != owner);
14          vm.prank(randomAddress);
15         string memory expectPassword = "myNewPassword";
16         passwordStore.setPassword(expectPassword);
17
18         vm.prank(owner);
19         string memory actualPassword = passwordStore.getPassword();
```

```
20          assertEq(actualPassword, expectPassword);
21      }
```

Recommended Mitigation: Add a access control to the `setPassword` function.

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

Informational

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

Description:

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3    @>  * @param newPassword The new password to set.
4        */
5       function getPassword() external view returns (string memory) {
```

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

Impact: The natspec is incorrect.

Recommended Mitigation:

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