Protocol Audit Report

Version 1.0

AzmaeenGH

December 15, 2023

Protocol Audit Report

AzmaeenGH

December 15, 2023

Prepared by: AzmaeenGH

Lead Security Researcher: - AzmaeenGH

Table of Contents

Protocol Summary

PasswordStore is a protol 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 AzmaeenGH's 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:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

In 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

I audited this codebase in line with the instructions provided by Updraft Cirriculam. Auditor AzmaeenGH spend 2 hours on this code, and used foundry commands and testing methodology to find the reported bugs in the code.

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

Descriptions: 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: (Proof of Code)

The below test case shows how anyone 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
```

4. You can then parse the hex result to a string value:

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

And then we 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 control, meaning a non-owner could change the password

Descriptions: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overall purpose of the smart contract is that `This function allows only the owner to set the new password`

```
1
2       function setPassword(string memory newPassword) external {
3   @>      // There are no access controls
4           s_password = newPassword;
5           emit SetNetPassword();
6       }
```

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

Proof of Concept:

Add this code to the PasswordStore.t.sol test file.

Code

```
1
2       function test_anyone_can_set_password(address randomAddress) public
            {
3           vm.assume(randomAddress != owner);
4
5           // the randomAddress is calling the setPassword() and setting
                the password
6           vm.prank(randomAddress);
7           string memory expectedPassword = "newPasswordX";
8           passwordStore.setPassword(expectedPassword);
9
10          // The owner calling the getPassword(), to check what password
                is stored
11          vm.prank(owner);
12          string memory actualPassword = passwordStore.getPassword();
13
14          /// Confirming that the randomAddress has succeded in setting
                the password,  which (according to required logic) it should
                 not be able to.
15          assertEq(expectedPassword, actualPassword);
16      }
```

Recommended Mitigation: Add an access control modifier 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 doesnt exist, causing the natspec to be incorrect

Descriptions:

```
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: Remove the incorrect natspec line.

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