Protocol Audit Report

Version 1.0

Axon.io

July 7, 2024

Protocol Audit Report

Adebayo Halir Shola

July 7, 2024

Prepared by: Halir Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of user's passwords. It is designed to be used by a single user and only the owner should be able to set and access tis password.

Disclaimer

Adebayo Halir Shola 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 corresponds to the folllowing 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

Hours spent, tools used 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 data off the chain below

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

Proof of Concept:

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

1. Create a locally running chain

```
make anvil
```

2. Deploy contract to the chain

```
make deploy
```

3. Run storage tool

1 is used because that's the storage slot of `s_password` in the contract.

```
cast storage <ADDRESS_OFCONTRACT> 1 --rpc-url http://localhost:8545
```

Output(hex):

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

convert hex to string

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

Recommended Mitigation: Due to this the overall architecture of the contract should be rethought, one could encrypt the password offchain then store this on the chain. This would require user to remember another password offchain to decrypt the password. However, you'd also likely want to remove the view function as you won'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 control meaning a non-owner could change the password

Description: The `PasswordStore::setPassword` is set to be `extenal` function, however, the natspec of the function and overall purposer of the contract is that `This function alows only the owner to set a new password.`

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

Impact: Anyone could change thr password of the contract, severly breaking the contract intended functionality

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

Code

```
 function test_non_owner_can_password(address randomAddress) public {
        vm.prank(randomAddress);
        string memory expectedPassword = "randomPassword";
        passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation: Add an access control conditional to the set `setPassword` function

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

Likelihood & impact - Impact: HIGH - Likelihood: HIGH - Severity: HIGH

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist hence the natspec is incorrect

Description:

```
 /*
    * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
*/

    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.
```

Likelihood & impact

- Impact: None
- Likelihood: High
- Severity: Informational/Gas/Non-crits