Protocol Audit Report

Version 1.0

Cyfrin.io

June 19, 2024

Protocol Audit Report

Mohammad Ahadinejad

March 7, 2023

Prepared by: [Mohammad Ahadinejad] Lead Security Researcher:

- Mohammad Ahadinejad

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 YOUR_NAME_HERE 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

commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Gas Optimizations	0
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visibile to anyone, therefore it's not private anymore

Description: Any Data that stored in blockchain is visible by anyone. making data public or private just specify the accibility through other functions and contracts not visibility. The `PasswordStore::s_password` variable is intented to be a private variable and only accessed through the

`PasswordStore::getPassword` function, which is intented to be only called by the owner of the contract.

Impact: Anyone can read the private password, which leads to severly breaking the functionality of the contract.

Proof of Concept: The bellow test case shows how anyone can read the password directly from the blockchain.

1. First make the anvil local network in bash terminal.

```
anvil
```

2. Then in another bash terminal deploy the `DeployPasswordStore.s.sol` using the command bellow.

```
make deploy
```

3. Due to the fact that we know `PasswordStore::s_password` variable is stored in slot 1 of the contract we can access the byte version of the variable using code bellow.

```
cast starage {contract address of deployed contract} 1 --rpc-url {
    address of local network (e.g. http://127.0.0.1:8545)}
```

4. Now we only need to convert last command output from byte to string

```
cast parse-bytes32-string {bytes}
```

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 dycript 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 dycripts your password.

[H-2] `PasswordStore::setPassword` function has no access control, therefore anyone can change the password

Description: The `PasswordStore::setPassword` is an external function which natspec of the function is that `This function allows only the owner to set a new password`

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

```
5       }
```

Impact: Anyone can set/change the password of the smart contract, severly beaking the contract intended functionality.

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

Code

```
1       function test_anyone_can_set_password(address randomAddress) public
            {
2           vm.assume(randomAddress != owner);
3           vm.prank(randomAddress);
4           string memory exoectedPassword = "myNewPassword";
5           passwordStore.setPassword(exoectedPassword);
6
7           vm.prank(owner);
8           string memory actualPassword = passwordStore.getPassword();
9           assertEq(exoectedPassword, actualPassword);
10      }
```

Recommended Mitigation: Add an access controll condition to the `PasswordStore::setPassword` function

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

Informational

[I-1] The `PasswordStore::getPassword` natspac indicates a parameter that doesn't exist, causing the natspac 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
6       function getPassword() external view returns (string memory) {
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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