Protocol Audit Report

Version 1.0

gorgut

July 4, 2024

Protocol Audit Report

Gorgut

June 3, 2024

Prepared by: AlexGorgut Lead Auditor: - Alex Gorgut

Table of Contents

Protocol Summary

The PasswordStore contract aims to allow the owner to securely store and retrieve a private password. It provides functions for setting and getting the password, but only the owner is authorized to access it.

Disclaimer

The Gorgut 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
Likelihood	High	H	H/M	M
	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

Scope

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

- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

Roles

- Owner: The user can set the password and read the password.
- Outsides: No one else should be able to set or read the password. # Executive Summary The PasswordStore contract aims to provide basic password storage for its owner. However, critical vulnerabilities render it insecure and unsuitable for use in its current state.

Issues found

Severtity | 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 intented to be private variable and only accessed through the `PasswordStore::getPassword` function, which is intented to be only called by the owner of the contract.

Impact: Changing visibility of variables only applies to smart contracts calls and not blochcain visibility.

Proof of Concept:

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 <CONTRACT_ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You will get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000000000000000`

You can then parse that hex to a string with:

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

And get an output of: myPassword

Recommended Mitigation: Do not store passwords in plain text. Instead, securely hash the password using a strong cryptographic hash function like keccak256 and store only the hash. When verifying a password, hash the user's input and compare it to the stored hash.

[H-2] `PasswordStore::setPassword` has no access controls, meaning non-owner user could change the password.

Description: Despite the NatSpec comment specifying `This function allows only the owner to set a new password`, the `external PasswordStore::setPassword` function lacks access control, allowing anyone to call it.

```
    function setPassword(string memory newPassword) external {
@>    // @audit Anybody can call the function setPassword. There are no
    access controlls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Externall address can call the `setPassword` function and set/change the password, breaking the contract intended functionality.

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

Code

```
    function test_any_user_call_setPassword(address randomAddress)
       public {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);
```

```
6
7             vm.startPrank(owner);
8         string memory actualPassword = passwordStore.getPassword();
9         assertEq(actualPassword, expectedPassword);
10    }
```

Then call this test `bash/zsh forge test --match-test test_any_user_call_setPassword`

Recommended Mitigation:
To fix this vulnarability you have to add onlyOwner access controll checker. The same that been used in `PasswordStore::getPassword` function:

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

Informational

[I-1] `PasswordStore::getPassword` has no parameters to pass, even so nat spac

Description:

```
1   /*
2       * @notice This allows only the owner to retrieve the password.
3       // @audit there is not newPassword parameter!
4       * @param newPassword The new password to set.
5       */
6           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.
```