PasswordStore Audit Report

Version 1.0

0xl33

December 7, 2024

PasswordStore Audit Report

0xl33

December 6, 2024

Prepared by: 0xl33

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

0xl33 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 0xl33 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

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

Owner: 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
Informational	1
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone, making it 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 any data off chain below.

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

Proof of Concept: (Proof of Code)

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

The `makeAddr` helper function is used to setup an `attacker` address to call the `setPasword()` function:

```
contract PasswordStoreTest is Test {
    PasswordStore public passwordStore;
    DeployPasswordStore public deployer;
    address public owner;
+   address public attacker;

    function setUp() public {
        deployer = new DeployPasswordStore();
        passwordStore = deployer.run();
        owner = msg.sender;
        // attacker address
+       attacker = makeAddr("attacker");
    }
}
```

The following test, sets the password to `"attackerPassword"` using the attacker address. When run, this test will pass, demonstrating that the attacker can set the password:

```
    function test_poc_non_owner_set_password() public {
        // initiate the transaction from the non-owner attacker address
        vm.prank(attacker);
        string memory newPassword = "attackerPassword";
        // attacker attempts to set the password
        passwordStore.setPassword(newPassword);
        console.log("The attacker successfully set the password:"
            newPassword);
    }
```

Run the test:

```
forge test --mt test_poc_non_owner_set_password -vv
```

Which yields the following output:

```
unning 1 test for test/PasswordStore.t.sol:PasswordStoreTest
[PASS] test_poc_non_owner_set_password() (gas: 20776)
Logs:
  The attacker successfully set the password: attackerPassword

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.36ms
```

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` function has no access controls, meaning anyone can change the password

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

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

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

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

Code

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

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

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

Medium

None.

Low

None.

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec 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      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.
```