Protocol Audit Report

Version 1.0

github.com/z0Ldev

May 6, 2024

Protocol Audit Report

z0L

May 6, 2024

Prepared by: z0L Lead Researcher: - z0L

Table of Contents

Protocol Summary

PasswordStore is a protocol that allows users to store and retrieve their passwords securely. The contract ensures that only the owner can access and retrieve it.

Disclaimer

The z0L 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

The findings described in this document correspond the following 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

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 any chail 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 can read the password directly from the blockchain.

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

Youll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: To address this issue, consider encrypting the password off-chain before storing it on-chain. This way, even if the encrypted password is visible on-chain, it would be meaningless without the decryption key. The decryption key should be kept securely off-chain by the user. Additionally, remove the view function to prevent accidentally exposing the decryption key in a transaction.

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

[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, implies that `this function is intended to be called only by the owner of the contract`. Without any access controls, any user can call this function and change the stored password.

```
function setPassword(string memory newPassword) external {
@>  // @audit - There are no access controls to ensure only the owner
    can call this function
     s_password = newPassword;
    emit SetNetPassword();
}
```

Impact: Any user can call the `setPassword` function and change the stored password, breaking the core functionality of the contract.

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

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 to ensure only the owner of the contract can call it. This can be done by checking the `msg.sender` against the owner address:

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

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

Informational

[I-1] The `PasswordStore::getPassword` natspec comment is incorrect. It indicates a parameter when there are none.

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 comment is incorrect and may mislead developers about the function's parameters.

Recommended Mitigation: Remove the incorrect natspec line.

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

Likelihood & Impact: - Impact: NONE - Likelihood: HIGH? - Severity: Informational/Gas/Non-crits