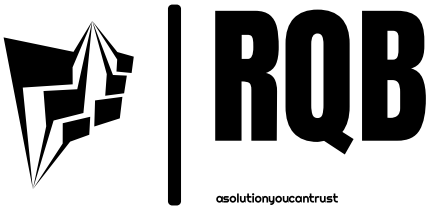


# Protocol Audit Report

Version 1.0

*Roqbell*

January 16, 2024

# Protocol Audit Report

Roqbell

January 10, 2024

Prepared by: Roqbell Lead Security Researchers: - xxxxxxx

## 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 it is not designed to be used by multiple users. Only the owner should be able to store a password and then retrieve it later. Others should not be able to access the password.

## Disclaimer

The Sec Research 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

**The finding described in this document correspond with the following commit hash:**

```
- Commit Hash:  2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

### Scope

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

### Roles

- Owner: THe user who can set and read the password.
- Others: No one else to be have access to store or read the password.

### Executive Summary

*How the audit went?, Type of thigs you found, etc. We spent X hours with Y auditors using Z tools. etc*

### Issues found

| Severity | Numbers of issues found |
|---|---|
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Info | 1 |
| Total | 3 |

### Findings

#### [H-1] Storing password on-chain makes it visible to anyone & 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 called only by the owner of the contract.

We show one such method of reading data off-cahin 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

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make anvil
```

3. Run the storage tool

We use 1 because that's the storage slot of `s_password` in the contract.

```
cast storage <CONTRACT_ADDRESS> 1 --rpc-url http://127.0.0.1:8485
```

You'll get an output that looks like this:

`0xblablabla`

You can then parse the hex to a string with:

```
cast parse-bytes32-string 0xblablabla
```

And get an otput of:

```
myPassword
```

**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 decypt your password.

### [H-2] `PasswordStore::setPassword` hass no access control, 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 the overall purpose of the smart contract is that `This function allows only the owner to set a new password.`

```
    function setPassword(string memory newPassword) external {
@>      // @audit - there are access control
        s_password = newPassword;
        emit SetNetPassword();
```

```
5       }
```

**Impact:** Anyone can set/change the password of the contract, severly breaking the contract 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           vm.prank(owner);
7           string memory actualPassword = passwordStore.getPassword();
8           assertEq(actualPassword, expectedPassword);
9       }
```

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

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

### High

### Medium

### Low

### Informational

#### [I-1] The `PasswordStore::getPsasword` 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 say it should be `getPassword(string)`.

**Impact:** The Natcpec is incorrect.

**Recommended Mitigation:** Remove the incorrect natspec line.

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

### Gas