Protocol Audit Report

Version 1.0

Seeleon

July 26, 2025

Protocol Audit Report

Seeleon

July 26, 2025

Prepared by: Seeleon Lead Researcher: - Seeleon

Table of Contents

Protocol Summary

Protocol stores an user's password in a blockchain and can be changed.

Disclaimer

The Seeleon 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

```
#--- 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

** I spent 2 hours using Foundry tool for auditing purposes. **

Issues found

Severity	Numbers 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 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.

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 slof of `s_password` in the contract.

```
cast storage <ADDRESS HERE> 1 --rpc-url 127.0.0.1:8545
```

You'll 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 outpu of:

```
myPassword
```

Recommended Mitigation: Do not store plaintext passwords on-chain. Instead, store a keccak256 hash of the password, and verify inputs by comparing their hash. If password recovery is needed, handle encryption and decryption off-chain.

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 smart contract is that `This function allows only the owner to set a new password`.

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

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

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.

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

```
3 }
```

Likelihood & Impact:

Informational

Likelihood & Impact: ore::getPassword‘ natspec indicates a parameter that doesn’t exist, causing the natspec to the incorrect.

Description:

```
1     /*
2      * @notice This allows only the owner to retrieve the password.
3      //@audit there is no 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()` which 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.
```

Likelihood & Impact:

Gas