Protocol Audit Report

sklnhunt

June 22, 2025

Protocol Audit Report

Version 1.0

sklnhunt

June 22, 2025

Protocol Audit Report

sklnhunt

June 22, 2025

Prepared by: Confidential Lead Security Researcher: - sklnhunt

Table of Contents

Protocol Summary

A smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The Krunal 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

Commit Hash:

`2e8f81e263b3a9d18fab4fb5c46805ffc10a9990`

Scope

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

Roles

- Owners
- Outsiders

Executive Summary

- The audit was successful. The codebase size is small and we are managed to identify HIGH and Informational severity vulnerabilities. We have spent 2 hours on this code base.

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

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, severly breaking the functionality of the protocol.

Proof of Concept: (Proof of Code)

The below test case shows how anyone can read the password 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
```

You will get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can parse that hex to a string with:

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

And get an output 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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

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 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

```
    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 access control conditional to the `setPassword` function.

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

Likelihood & Impact:

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

Informational

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

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 say it should be `getPassword(string)`.

Impact: The natspec is incorrect.

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

Report generated by: sklnhunt
Date: June 22, 2025**Confidentiality Level:** Internal/Client Only