Protocol Audit Report

Version 1.0

Geekybot

March 28, 2025

Protocol Audit Report

geekybot

March 27, 2025

Prepared by: Geekybot Lead Auditors: - Utpal Pal

#sda

Table of Contents

Protocol Summary

A smart contract applicatoin 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. This protocol is intended for used by one user for each contract.

Disclaimer

The Geekybot 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 findings described in this document corresponds to the commithas 226e6a4a53fc5021695b67c16de5501ec from the repository 3-password-store-audit

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

We sepent alround 8 hours on this codebase in scope, we found 2 `HIGH` severity issues, with one `INFO` level issue in this scope. We used `foundry` and `solidity-metrics` tools to explore this.

Issues found

Severity	No of Issues Found
HIGH	2
MEDIUM	0
LOW	0
INFO	1
TOTAL	3

Findings

High

[H-1] Storing the password on-chain is always visible to anyone, it is no longer private

Description: All data stored on chain is visible to anyone, and can be read by anyone directly from the blockchain. The `PasswordStore::s_password` variable is intended to be private variable and only be accessed by owner by calling `PasswordStore::getPassword` function.

Impact: Since anyone can read the password, this essentially break the protocol functionality

Proof of Concept: 1. Run a local chain

```
1 anvil
```

2. Run the deploy script

```
make deploy
```

3. Inspect the storage using `forge`, we use storage 1 as it is the storage for `PasswordStore::s_password`. contract address after deploying is `0x5FbDB2315678afecb367f032d93F642f64180aa3`

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 -r http://127.0.0.1:8545
```

4. We will find a value like `0x6d7950617373776f726400000000000000000000000000000000000000000000`
now we parse this to string using

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

this will return value `myPassword`

Recommended Mitigation: This whole architecture is flawed, one way to do this would be encrypting the password off-chain with another password that user has to remember, but beaware to use the getPassword view function as user could send the decrypting password as a transaction parameter which will be again visible by anyone directly reading from blockchain.

[H-2] Missing access control at `PasswordStore::setPassword` function, meaning any non-owner can set/update the `PasswordStore::s_password` variable

Description: Missing access control at `PasswordStore::setPassword` function would result in anyone able to set password, which breaks the intended functionality of the protocol as `This function allows only the owner to set a new password` severly impacting the protocol.

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

Impact: Anyone can set the password of this contract, severly breaking the purpose of the contract

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

Code

```
function test_anyone_can_set_password(address randomUser) public {
        vm.assume(randomUser != owner);
        vm.startPrank(randomUser);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);
        vm.stopPrank();
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

and run

```
forge test --mt test_anyone_can_set_password
```

Recommended Mitigation: Add an access control condition to the `PasswordStore::setPassword` function

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

Informational

[I-1] The `PasswordStore::getPassword` function indicates a parameter that does not exist, causing the natspec 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 incorrect, according to natspec `getPassword(string)` which should be `getPassword()`

Impact Incorrect natspec

Recommended Mitigation: Remove the incorrect natspec

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