Protocol Audit Report

Version 1.0

Cyfrin.io

May 15, 2025

Protocol Audit Report

adebisivince

May 15, 2025

Prepared by: Cyfrin Lead Security reserachers: - adebisivince

Table of Contents

Protocol Summary

Password store is a protocol dedicated to store and retieve a users passwords. the protocol is designed to be used by a single user and not multiple users. Only the owner should be able to set and access this password.

Disclaimer

The adebisivince 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

`2e8f81e263b3a9d18fab4fb5c46805ffc10a9990`

Scope

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

Roles

- Owner: The user who can set the password and read the password.
- User: No one else should be able to set or read the password.

Executive Summary

We spent 2 hours using foundry to audit the code.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Gas	0
Total	3

Findings

High

[H-1] Storing the password onchain 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 called by the onlyOwner of the contract.

Impact: Anyone can read the private password, severly breaking the functionality of the protocol.

Proof of Concept: The below test case shows anyone can read the password directly from the blockchain 1. create a locally running chain

```
make anvil
```

2. deploy the contract to the local 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 [deployed_contract_address] [storage_slot] --rpc-url
↪  http://127.0.0.1:8545
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
↪  http://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 0x6d7950617373776f726400000000000000000000000000000000000000000
```

And you'll get the output:

```
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 passwor. However, you'd also likely want to remove the view function as you wouldnt want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has 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 overall purpose of the smart contract is that `this function allows only owner to set the ne password`

```
function setPassword(string memory newPassword) external {
@> //@audit - there are no access control
        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 random_address) public {
        vm.assume(random_address != owner);
        vm.prank(random_address);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

Recommended Mitigation: Add an access control conditioner to the `sPasswordStore::setPassword` function to ensure only the owner can set the password.

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

Informational

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

Description: The `PasswordStore::getPassword` function is signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

Impact: the natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line

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