Protocol Audit Report

Version 1.0

Cyfrin.io

August 29, 2024

Protocol Audit Report

Cyfrin.io

August 27, 2024

Prepared by: IconArt Lead Security Research: - Oke Abdulquadri

Table of Contents

Protocol Summary

Password Store is a protocol dedicated to store and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access the password.

Disclaimer

The IconArt 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

The findings described in this document correspond the following commit hash: Commit Hash:

```
d84b95c02ca47e5d45300c62df1b2d625ae9f727
```

Scope

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

Roles

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

Executive Summary

The audit covered the entire lifecycle of the smart contract, including code review, vulnerability assessment, functional testing, and gas optimization analysis. Our team thoroughly examined the contract for common vulnerabilities, such as access control issues, and potential logic flaws.

Issues found

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

Findings

High

[H-1] Password stored on-chain is visable by anyone, not matter the solidity visibility variable

Description: All data stored on-chain is visible by anyone, and it can be directly retrieved for the EVM storage. The `passwordStore::s_password` is intended to be a private variable, and it should

only be accessed through the `passwordStore::getPassword` function, which is intended to be called only be the owner of the contract.

However, anyone can directly read the date from EVM storage using any number of off-chain methodologies.

Impact: The password is not private.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use the foundry-cast tools to directly read from the storage of the contract, without being the owner.

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> --rpc-url http://127.0.0.1:8545
```

You can 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 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 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 decrypts your password.

[H-2] `PasswordStore::setPassword` has no access control, meaning non-owner could change the password.

Description: The `Password::setpassword` function is set to be an external function, however the natspec of this function and the overall purpose of the smart contract is that `This function allow only the owner to set new 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, severely this break the contract functionality.

Proof of Concept: Add the following to the `passwordStore.t.sol` test file.

Code

```
    function test_anyoneCanSetPassword(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(expectedPassword, actualPassword);

    }
```

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

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

Informational

[S-#] The `PasswordStore::getPassword` natspec indicate a parameter that does not exist, which makes the natspec incorrect.

Description: The natspec indicate that `getPassword` function should take in a `newpassword` as parament, but the function doesn't hold any parameter.

Impact: It does not have any impact though.

Recommended Mitigation The following comment should be removed from `line 36`

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