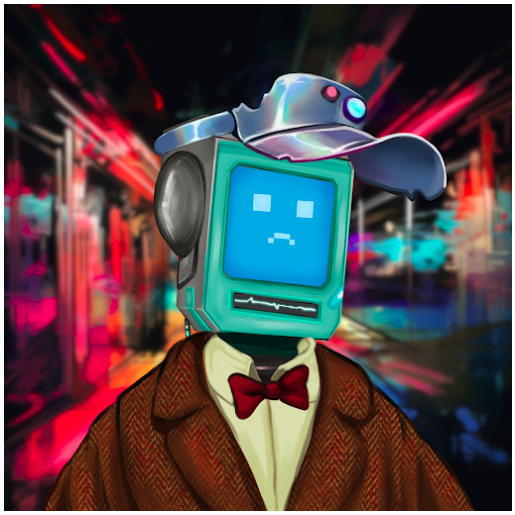

Protocol Audit Report

Version 1.0

Cyfrin.io

November 24, 2024

Protocol Audit Report

Anon :3

March 7, 2023

Prepared by: Anon :3 Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

The protocol PasswordStorage is a protocol thats designed to store a password on-chain, to be retrieved by the owner anytime they want, It’s desgigned so that only the owner is able to access that password.

Disclaimer

The YOUR_NAME_HERE 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

”’ 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990 ”’

Scope

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

Roles

- Owner: The user who can set and retrive a password
- Outsider: Any other users who will not be able to set or retrieve a password set by the owner

Executive Summary

How it went?

Issues found

Severity	Number 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 (Root cause + Impact)

Description: All data stored on-chain is visible to anyone, and can be read directly by anyone The `PasswordStore::s_password` variable is intented to be private and only accessable by the `PasswordStore::getPassword` function only owner can run

We show one such method of reading 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 directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract on the chain

```
make deploy
```

3. Get the storage, This will return everything stored in storage, which includes the password

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

Number 1 after contract address is where password is stored inside the storage you will get output like this: `0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. Grab the hex value of the password, and convert it into string

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

You will get output: myPassword

Recommended Mitigation: Due to this, overall architecture of the contract should be rethough, one could encrypt the password off-chain and store the encrypted password in storage, and it gets decrypted when owner gets the password off-chain, however, you'd also lickely want to remove the view function as you wouldn't want the user to accidentally send a transaction witht he password that decrypts your password.

[H-2] `PasswordStore::setPassword` function has no access controls, meaning a non-owner can change the password (Root Cause + Impact)

Description: The function `PasswordStore::setPassword` is set as an external function, and it does not have access control, However, the natspec of that function clearly states that `This function allows only the owner to set a new password`. wether or not the owner is calling the function.

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

Impact: Non-owner users can call the function `PasswordStore::setPassword`, and change the password, severly breaking the contracts intended functionality.

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

Code

```
    function testAnyOneCanSetPassword(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();
}
```

Informational

[I-1] The `PasswordStore::getPassowrd` natspec indicated a parameter that doesn't exist, causing the natspec to be wrong (Root Cause + Impact)

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) {
```

natspec indicated a param `newPassword`, but the function signature is `getPassword()`, not `getPassword(string)`

Impact: THe natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

```

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