Protocol Audit Report

Version 1.0

Cyfrin.io

October 21, 2024

Protocol Audit Report

Cyfrin.io

nov 20, 2024

Prepared by: Cyfrin Lead Auditors:

- Joran Vanwesenbeeck

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage 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 this 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
	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
- In Scope:

Scope

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

Roles

Owner: Is the only one who should be able to set and access the password. For this contract, only the owner should be able to interact with the contract.

Executive Summary

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.

Description: All data stored on-chain is visibe to anyone, and can be read directly from the blockchain. the `PasswordStore::s_password` variable is intended to be a private variable wich only the owner can access true the `PasswordStore::getPassword` function.

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 cnan read the password directly 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 sloth where `PasswordStore::getPassword` gets saved.

```
cast storage <CONTRACT_ADDRESS_HERE> 1 --rpc-url <RPC_URL>
```

you'll get an output that looks like this: 0x6d7950617373776f726400000000000000000000000000000000000000000014

You can then parse that hex to a string with:

```
cast parse-byte32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

and you an output of:

```
myPassword
```

Recommended Mitigation: As a result, the overall architecture of this contract needs reconsideration. One approach could be to encrypt the password off-chain, and then store the encrypted version on-chain. This would require the user to remember an additional off-chain password to decrypt the password. Additionally, it would be advisable to remove the view function to prevent users from inadvertently sending a transaction that reveals the decryption password.

[H-2] `PasswordStore::setPassword` has no access control, giving everyone access to the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function without any access control, however, the natspec and overall purpose of this smart contract is `This function allows only the owner to set the password`.

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

Impact: Anyone can access/change the password of the contract, severly breaking the purpose of this contract.

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

Code

```
  function test_anyone_can_set_password(address randomAddress) public {
        string memory expectedPassword = "myNewPassword";
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        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 != owner){
    revert PasswordStore_notOwner();
}
```

[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()` which 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.
```