Protocol Audit Report

Version 1.0

0xPhilz

January 2, 2024

Protocol Audit Report

0xPhilz

December 31, 2023

Prepared by: 0xPhilz Lead Auditors: - 0xPhilz

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Repo URL
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
  - [H-1] Storing the password on-chain makes it visible to everyone, and no longer private
  - [H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password
- Informational
  - [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Protocol Summary

PasswordStore is a protocol 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.

Disclaimer

The 0xPhilz 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 audit was based on the commit hash given below; 7d55682ddc4301a7b13ae9413095feffd9924566

Repo URL

https://github.com/Cyfrin/3-passwordstore-audit

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 or read the password.

Executive Summary

The audit was carried out by a single auditor and the following tools were utilized; solidity metrics, cloc, foundry. A total of 5 hours was spent on the audit and 3 findings were made; 2 highs and 1 informational.

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 everyone, and no longer private

Description: All data stored on chain is public and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function.

I show one such method of reading any data off chain below.

Impact: Anyone is able to read the private password, severly breaking the functionality of the protocol.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly 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> 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 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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

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

Description: `The PasswordStore::setPassword` function is set to be an external function, however the purpose of the smart contract and function's natspec indicate that `This function allows only the owner to set a new password`.

```
    function setPassword(string memory newPassword) external {
        // @Audit - There are no Access Controls.
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Impact: Anyone can set/change the stored password, severly breaking the contract's intended functionality

Proof of Concept: The below test case shows that any user can set the password

Code

```
function test_any_user_can_set_password(address randomUser) public {
        vm.assume(randomUser != owner);
        vm.prank(randomUser);
        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 `PasswordStore::setPassword` function

```
if (msg.sender != 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:

```
    /*
     * @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 natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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