Protocol Audit Report

Version 1.0

0xPicture

January 4, 2024

Protocol Audit Report

0xPicture

January 4, 2023

Prepared by: [0xPicture] Lead Auditors:

- 0xPicture

Table of Contents

Protocol Summary

“This protocol is designed to securely store user passwords and provide them with access to their credentials. Only the account owner has the right to access their passwords. This ensures enhanced security and prevents unauthorized access. The protocol maintains the confidentiality of login information while offering easy and secure accessibility to the legitimate user.”

Disclaimer

0xPicture 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

Scope

The scope of this audit was to review the security and functionality of the PasswordStore smart contract, with a particular focus on the storage and access control mechanisms for passwords.

Roles

The roles involved in this audit included security auditors responsible for analyzing the smart contract code, identifying vulnerabilities, and suggesting mitigations.

Executive Summary

This audit revealed critical vulnerabilities in the PasswordStore smart contract, particularly concerning the storage and modification of passwords. The issues identified pertain to public visibility of stored passwords ([H-1]) and the lack of access controls on the setPassword function ([H-2]). These vulnerabilities pose a high risk to the security and intended functionality of the protocol.

Issues found

Two high-severity issues were identified:

1.Public Visibility of Stored Passwords ([H-1]) **Description:** All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore : s_password` variable is inteded to be a priavet variable and only accessed through `PasswordStore::getPassword` function whish is supposed to call only by the contract owner

Impact: Anyone can read the pasword, whish break the fonctionality of the protocol

Proof of Concept:

The below test case shows how anyone can read the password directly from the blockchain.

1. Creat 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 `PasswordStore::s_password`

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 rpc-url
    127.0.0.1:8545
```

You'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000000`

cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000014

And get an output of :

```
myPassword
```

Recommended Mitigation:

Use Encryption: Encrypt sensitive data like passwords before storing them on the blockchain. This way, even if someone reads the data, they won't be able to understand it without the decryption key.

2.Lack of Access Controls on setPassword ([H-2]) `PasswordStore::setPassword` is external and can be called by anyone, so every user can call any password stored by the protocol. According to the comment below the function PasswordStore::setPassword it has to be only the owner who can access to this information

```
function setPassword(string memory newPassword) external {
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/get any password, whish break the inteded functionality of the protocol

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

```

function test_anyone_can_set_password(address randomeAddress) public {
        vm.assume(randomeAddress != owner);
        vm.prank(randomeAddress);
        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();
}
```

Findings

Severity	Number of findings
High	2
Medium	0
Low	0

Severity	Number of findings
Informational	0
Gas	0