Protocol Audit Report

Version 1.0

Vkgoud

February 28, 2025

Protocol Audit Report

Vk goud

Feb 28, 2025

Prepared by: Vkgoud

Lead Auditors:

- Vamshi Krishna Goud

Table of Contents

Protocol Summary

Roses are red, violets are blue, use this DatingDapp and love will find you.! Dating Dapp lets users mint a soulbound NFT as their verified dating profile. To express interest in someone, they pay 1 ETH to "like" their profile. If the like is mutual, all their previous like payments (minus a 10% fee) are pooled into a shared multisig wallet, which both users can access for their first date. This system ensures genuine connections, and turns every match into a meaningful, on-chain commitment.

Risk Classification

MEDIUM

Audit Details

Scope

src/ #– LikeRegistry.sol #– MultiSig.sol #– SoulboundProfileNFT.sol

Compatibilities

Chains:

Ethereum/EVM Equivalent Tokens:

ERC721 standard

Medium

[M-1] - Reentrancy Risk in mintProfile Due to External Call Before State Update

Description: The mintProfile function in the `SoulboundProfileNFT` contract calls `_safeMint(msg.sender, tokenId)` before updating critical state variables such as `_profiles[tokenId]` and `profileToToken[msg.sender]`. Since `_safeMint` invokes `IERC721Receiver(to).onERC721Received`, if `msg.sender` is a contract, it can execute arbitrary code before the function completes. This allows potential reentrant calls that could manipulate state inconsistencies or execute unintended logic.

Impact:

1. If a malicious contract is used as `msg.sender`, it can re-enter the `mintProfile` function via `onERC721Received`.
2. This could lead to double minting or inconsistent state, where `profiles` and `profileToToken` are not correctly updated.
3. The vulnerability could potentially allow an attacker to mint multiple NFTs or bypass profile uniqueness constraints.

Proof of Concept:

```
/// @notice Mint a soulbound NFT representing the user's profile.
    function mintProfile(string memory name, uint8 age, string memory
        profileImage) external {
        require(profileToToken[msg.sender] == 0, "Profile already
            exists");

        uint256 tokenId = ++_nextTokenId;
            // Store metadata first (before external calls)

+         _profiles[tokenId] = Profile(name, age, profileImage);
+         profileToToken[msg.sender] = tokenId;

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

    //  External call happens after state update

        _safeMint(msg.sender, tokenId);

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

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

Recommended Mitigation:

1. Follow the Checks-Effects-Interactions - (CEI) pattern to ensure state updates occur before external calls.
2. Move `_safeMint` after all state changes to prevent reentrancy risks.
3. By consider using OpenZeppelin's `ReentrancyGuard` to prevent reentrant calls ("https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol"