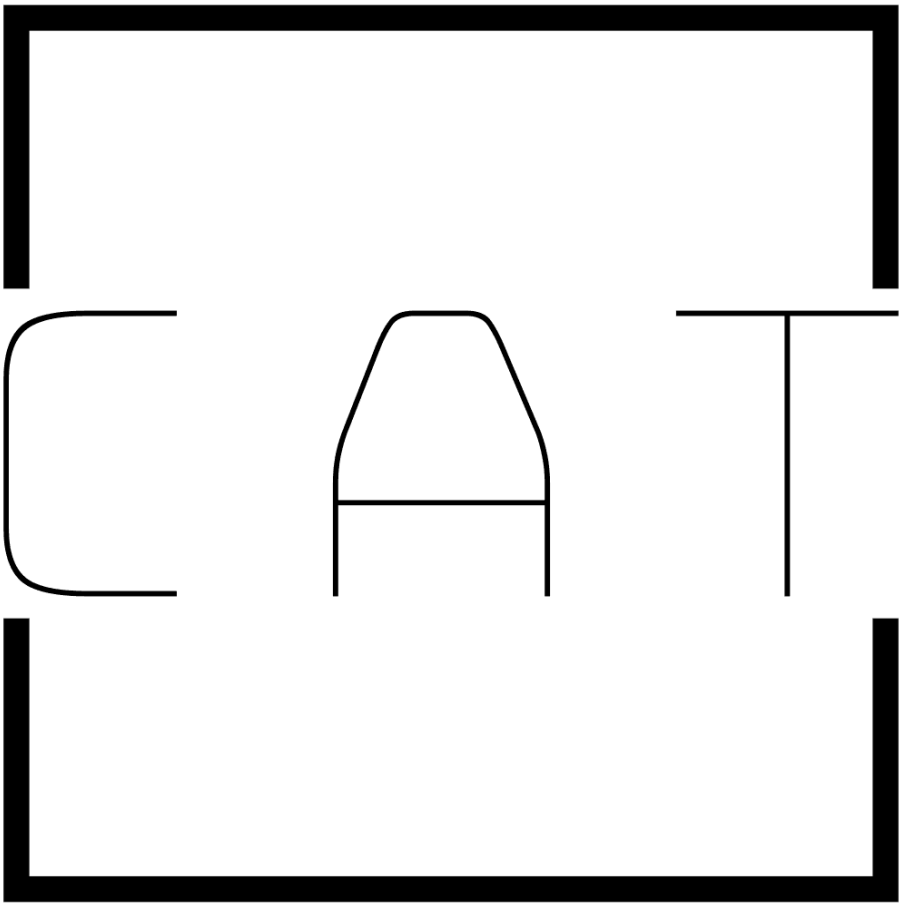

Protocol Audit Report

Prepared by: Cat

Table of Contents

Protocol Summary

The Dating Dapp Protocol 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.

Disclaimer

Me and my 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

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	1
Medium	1
Low	1
Info	8
Total	11

Findings

High

[H-1] `_baseURI` not implemented.

Description: Without _baseURI(), the NFT metadata would not be correctly formatted as a valid data URI, and it wouldn't display properly. If _baseURI() is missing or incorrect, the metadata returned by tokenURI() won't have the correct data: scheme.

Recommended Mitigation: Add the following function in `SoulboundProfileNFT`

```
    function _baseURI() internal pure override returns (string memory) {
        return "data:application/json;base64,";
    }
```

Medium

[M-1] Potential reentrancy risk in `SoulboundProfileNFT::mintProfile`

Description: `SoulboundProfileNFT::mintProfile` performs a _safeMint operation before updating the contract's internal state variables. The _safeMint function from OpenZeppelin's ERC721 implementation invokes _checkOnERC721Received, which allows a receiving contract to execute arbitrary logic via the onERC721Received callback. If msg.sender is a malicious smart contract, it could attempt to re-enter `SoulboundProfileNFT::mintProfile` before the state variables are updated.

Recommended Mitigation: Reorder the state updates to occur before calling _safeMint. This ensures that critical data is set before any external interaction. Although no immediate exploit exists, following the principle of "Checks-Effects-Interactions" enhances the contract's resilience against potential attacks.

```
    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);
    }
```

Low

[L-1] Local variable shadowing in `SoulboundProfileNFT::mintProfile`.

Description: OpenZeppelin's ERC721 contract includes a public function name(), which returns the token collection's name. This function is accessible within `SoulboundProfileNFT` unless a local variable or parameter with the same name is declared. The function parameter "name" shadows ERC721.name(), meaning that inside `SoulboundProfileNFT::mintProfile`, "name" will always refer to the parameter. As a result, the use of "name" might be incorrect.

Recommended Mitigation: Change the "name" parameter to something else, like "profileName".

```
-   function mintProfile(string memory name, uint8 age, string memory
profileImage) external {
+   function mintProfile(string memory profileName, 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);
+       _profiles[tokenId] = Profile(profileName, age, profileImage);
        profileToToken[msg.sender] = tokenId;

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

Informationals

[I-1] No contract's balance checking before making funds transfer.

Description: There is no contract's balance checking before making the call in `MultiSigWallet::submitTransaction`. Adding an explicit check improves clarity and avoids unnecessary execution steps before the revert.

Recommended Mitigation: Add check before executing the transfer

```
    if (_to == address(0)) revert InvalidRecipient();
    if (_value == 0) revert InvalidAmount();
+   require(address(this).balance >= _value, "Insufficient balance");
```

[I-2] Unused Custom Error.

`MultiSigWallet::NotEnoughApprovals` is not used and should be removed

```
- error NotEnoughApprovals();
```

[I-3]: State variable changes but no event is emitted.

Add event in `LikeRegistry::withdrawFees`

```
+   event FeeWithdrawn(address indexed owner, uint256 amount);

    function withdrawFees() external onlyOwner {
        require(totalFees > 0, "No fees to withdraw");
        uint256 totalFeesToWithdraw = totalFees;

        totalFees = 0;
```

```
        (bool success,) = payable(owner()).call{value: totalFeesToWithdraw}("");
        require(success, "Transfer failed");

+       emit FeeWithdrawn(owner(), totalFeesToWithdraw);
    }
```

[I-4] Missing zero address checks.

Description: The `LikeRegistry` contract does not validate that the `_profileNFT` is not the zero address. This means that the `_profileNFT` could be set to the zero address, and fees would be lost.

Recommended Mitigation: Add a zero address check.

```
    constructor(address _profileNFT) Ownable(msg.sender) {
+       require(_profileNFT != address(0), "Invalid profile NFT address");
        profileNFT = SoulboundProfileNFT(_profileNFT);
    }
```

[I-5] Event is missing `indexed` fields.

Description: Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

- Found in src/MultiSig.sol [Line: 24]

```
    event TransactionCreated(uint256 indexed txId, address indexed to,
uint256 value);
```

- Found in src/MultiSig.sol [Line: 26]

```
    event TransactionExecuted(uint256 indexed txId, address indexed to,
uint256 value);
```

- Found in src/SoulboundProfileNFT.sol [Line: 25]

```
    event ProfileMinted(address indexed user, uint256 tokenId, string name,
uint8 age, string profileImage);
```

- Found in src/SoulboundProfileNFT.sol [Line: 26]

```
    event ProfileBurned(address indexed user, uint256 tokenId);
```

[I-6] Unchanged variables should be constant or immutable.

Immutable Instances:

```
LikeRegistry.profileNFT (src/LikeRegistry.sol#15) should be immutable
MultiSig.owner1 (src/MultiSig.sol#11) should be immutable
MultiSig.owner2 (src/MultiSig.sol#12) should be immutable
```

[I-7] Floating pragmas.

Description: Contracts should use strict versions of solidity. Locking the version ensures that contracts are not deployed with a different version of solidity than they were tested with. An incorrect version could lead to uninteded results.

Recommended Mitigation: Lock up pragma versions.

```
- pragma solidity ^0.8.19;
+ pragma solidity 0.8.19;
```

[I-8] Test Coverage.

Description: The test coverage of the tests are below 90%. This often means that there are parts of the code that are not tested.

```
| File                        | % Lines         | % Statements    | % Branches    |
% Funcs       |
|-----------------------------|-----------------|-----------------|---------------
|---------------|
| src/LikeRegistry.sol        | 0.00% (0/31)    | 0.00% (0/35)    | 0.00% (0/17)  |
0.00% (0/5)   |
| src/MultiSig.sol            | 0.00% (0/23)    | 0.00% (0/34)    | 0.00% (0/23)  |
0.00% (0/5)   |
| src/SoulboundProfileNFT.sol | 100.00% (23/23) | 96.55% (28/29)  | 55.56% (5/9)  |
85.71% (6/7)  |
| Total                       | 29.87% (23/77)  | 28.57% (28/98)  | 10.20% (5/49) |
35.29% (6/17) |
```

Recommended Mitigation: Increase test coverage to 90% or higher, especially for the Branches column.