Soulmate Audit Report

Version 1.0

Owen Lee.

February 29, 2024

Soulmate Audit Report

Owen Lee

Feb 29, 2024

Prepared by: [Owen Lee] Lead Auditors: - Owen Lee.

Table of Contents

Protocol Summary

The Soulmate Protocol offers a unique and rewarding experience for individuals seeking to connect with their soulmate. By minting a shared Soulbound NFT, earning LoveTokens, and participating in the staking contract, users can engage in a community-driven ecosystem centered around love and commitment. This protocol not only encourages meaningful relationships but also provides economic opportunities within its innovative framework.

Disclaimer

I, Owen Lee 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 me 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

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond to the following commit hash:

```
None.
```

Scope

- In Scope: (For this contest, just use the main branch)

Roles

- None.

Executive Summary

- The audit of the Soulmate Protocol revealed several key findings aimed at enhancing security and functionality. Notable recommendations include replacing the mint function with safeMint for improved transaction safety. Additionally, improvements to contract logic and event handling were advised to fortify the protocol against potential vulnerabilities.

Issues found

Severity	Number of issues found
Low	1
Medium	0
Low	0
Info	0
Total	1

Findings

Low

[L-1] `Soulmate::mintSoulmateToken` fails to validate recipient addresses in its `_mint` function, risking a DOS attack on the token.

Description: The attack vector described involves malicious actors exploiting the lack of recipient ad-dresss validation of the `onERC721Received` function within the `Soulmate::mintSoulmateToken` function, specifically through the use of the `_mint` function.

```
function mintSoulmateToken() public returns (uint256) {
        address soulmate = soulmateOf[msg.sender];
        if (soulmate != address(0)) {
            revert Soulmate__alreadyHaveASoulmate(soulmate);
        }

        address soulmate1 = idToOwners[nextID][0];
        address soulmate2 = idToOwners[nextID][1];
        if (soulmate1 == address(0)) {
            idToOwners[nextID][0] = msg.sender;
            ownerToId[msg.sender] = nextID;
            emit SoulmateIsWaiting(msg.sender);
        } else if (soulmate2 == address(0)) {
            idToOwners[nextID][1] = msg.sender;
            ownerToId[msg.sender] = nextID;
            soulmateOf[msg.sender] = soulmate1;
            soulmateOf[soulmate1] = msg.sender;

            idToCreationTimestamp[nextID] = block.timestamp;

            emit SoulmateAreReunited(soulmate1, soulmate2, nextID);

@>              _mint(msg.sender, nextID++);
        }

        return ownerToId[msg.sender];
    }
```

```

@> function _mint(address to, uint256 id) internal virtual {
        require(to != address(0), "INVALID_RECIPIENT");

        require(_ownerOf[id] == address(0), "ALREADY_MINTED");

        unchecked {
            _balanceOf[to]++;
        }

        _ownerOf[id] = to;

        emit Transfer(address(0), to, id);
    }
```

Impact: As a consequence of the attack, tokens sent to addresses incapable of handling them become irretrievably lost within the Ethereum blockchain. Since blockchain transactions are immutable, once tokens are sent to an address, they cannot be reversed or recovered thus resulting in a permanent loss of tokens from the token supply, adversely affecting `Soulmate` token holders and the overall ecosystem stability.

Proof of Concept:

1. A malicious actor may attempt to mint a `Soulmate` token by pairing a non-NFT compatible smart contract address with another address presumed to support NFT functionality.
2. A denial-of-service (DOS) attack transpires, resulting in the loss of the token within the Ethereum blockchain.

Proof Of Code

Place the following into the `SoulmateTest.t.sol`.

```
function test_NonNftContractrevertswhenMintingSoulmateContract() public
     {
        vm.prank(soulmate1);
        soulmateContract.mintSoulmateToken();

        assertTrue(soulmateContract.totalSupply() == 0);

        vm.prank(address(nonNftContractaddress));
        vm.expectRevert();
        soulmateContract.mintSoulmateToken();

// Both Soulmate 1 and the NonNftContract address dont have a token
    mainly because of the exploit in the NonNftContract.
        assertTrue(soulmateContract.totalSupply() == 0);
    }
}


contract NonNftContract {
//  Contract that doesnt implement the IERC721Receiver.onERC721Received
    ()
}
```

Recommended Mitigation: Use the `_safeMint` function instead of the `_mint` function since it carries out checks for the `onERC721Received` function which reverts back with the selector thus proving whether the recipient address is compatible to handle ERC-721 tokens or not.

```

-function _mint(address to, uint256 id) internal virtual {
-        require(to != address(0), "INVALID_RECIPIENT");

-       require(_ownerOf[id] == address(0), "ALREADY_MINTED");


-        unchecked {
-            _balanceOf[to]++;
-        }

-        _ownerOf[id] = to;

-        emit Transfer(address(0), to, id);
-    }


+function _safeMint(address to, uint256 id) internal virtual {
+        _mint(to, id);

+        require(
+            to.code.length == 0 ||
+                ERC721TokenReceiver(to).onERC721Received(msg.sender,
   address(0), id, "") ==
+                ERC721TokenReceiver.onERC721Received.selector,
+            "UNSAFE_RECIPIENT"
+        );
+    }



function mintSoulmateToken() public returns (uint256) {
        address soulmate = soulmateOf[msg.sender];
        if (soulmate != address(0)) {
            revert Soulmate__alreadyHaveASoulmate(soulmate);
        }

        address soulmate1 = idToOwners[nextID][0];
        address soulmate2 = idToOwners[nextID][1];
        if (soulmate1 == address(0)) {
            idToOwners[nextID][0] = msg.sender;
            ownerToId[msg.sender] = nextID;
            emit SoulmateIsWaiting(msg.sender);
        } else if (soulmate2 == address(0)) {
            idToOwners[nextID][1] = msg.sender;
            ownerToId[msg.sender] = nextID;
            soulmateOf[msg.sender] = soulmate1;
            soulmateOf[soulmate1] = msg.sender;
            idToCreationTimestamp[nextID] = block.timestamp;

            emit SoulmateAreReunited(soulmate1, soulmate2, nextID);

-            _mint(msg.sender, nextID++);
+            _safeMint(msg.sender, nextID++);
        }

        return ownerToId[msg.sender];
    }
```