Soulmate Audit Report

Version 1.0

FalseGenius

May 5, 2024

Soulmate Audit Report

FalseGenius

Soulmate 05, 2024

Prepared by: FalseGenius Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

Valentine’s day is approaching, and with that, it’s time to meet your soulmate!

We’ve created the Soulmate protocol, where you can mint your shared Soulbound NFT with an unknown person, and get LoveToken as a reward for staying with your soulmate. A staking contract is available to collect more love. Because if you give love, you receive more love.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	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

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

```
b3f9227942ffd5c443ce6bccaa980fea0304c38f
```

Scope

```
./src/
#-- Soulmate.sol
#-- LoveToken.sol
#-- Airdrop.sol
#-- Staking.sol
```

Roles

None

Executive Summary

I loved auditing this codebase. I spent 1~2 days using Foundry tests, Slither, Aderyn and Manual Review for the Audit

Issues found

Severity	Number of Issues Found
High	3
Medium	1
Low	1
Total	5

Findings

High

[H-01] Users without Soulmate NFT can claim rewards in `Airdrop::claim()`.

Description: As per documentation, the `Airdrop::claim()` function allows Soulmate NFT holders to claim loveToken. Due to lack of necessary validation checks, any user can claim loveTokens as if they had first minted Soulmate NFT by making use of default ID 0 returned by the state variable `Soulmate::ownerToId()`. If user is not in `ownerToId` mapping, default value of 0 is returned which allows them to get timestamp from `idToCreationTimestamp` and proceeding with the claim.

Additionally, there is a flaw in `soulmateContract.isDivorced()` check where msg.sender is the airdrop contract, so a divorced users can still claim.

Impact: Anyone lacking Soulmate NFT and divorced users can still mint loveTokens and claim rewards, breaking the intended functionality of the protocol.

Proof of Concept: Two soulmates get divorced after 7 days and they can still claim love tokens.

```
1
2     function testDivorcedUsersCanClaimLoveTokens() public
          soulmatesEntered {
3         vm.warp(block.timestamp + 7 days + 1 seconds);
4
5         vm.prank(soulmate2);
6         soulmateContract.getDivorced();
7
8         vm.prank(soulmate2);
9         airdropContract.claim();
```

```
10
11          assertEq(loveToken.balanceOf(soulmate2), 7 ether);
12      }
```

Recommended Mitigation: The following modifications are recommended.

- Consider modifying the `getDivorced()` to correctly handle msg.sender as a parameter

```
1
2   -   function isDivorced() public view returns (bool) {
3   -       return divorced[msg.sender];
4   -   }
5
6   +   function isDivorced(address sender) public view returns (bool) {
7   +       return divorced[sender];
8   +   }
```

- Start minting Soulmate NFT from nextID 1 onwards, so the ID of 0 will reserved for those that did not mint the NFT

```
1
2       function claim() public {
3
4   -       if (soulmateContract.isDivorced()) revert
        Airdrop__CoupleIsDivorced();
5   +       if (soulmateContract.isDivorced(msg.sender)) revert
        Airdrop__CoupleIsDivorced();
6
7   +       if (soulmateContract.ownerToId(msg.sender) == 0) revert();
8
9           ...
10          ...
11      }
```

[H-02] The function `Staking::claimRewards()` does not verify if a user has minted Soulmate NFT leading them to easily withdraw loveTokens from the vault.

Description: There are no checks in `claimRewards()` against users who did not mint soulmate NFT, since it is coded out with the assumption that anyone with a loveToken must have minted soulmate NFT. This allows attackers to take advantage of vulnerability in `Airdrop::claim()` to claim loveTokens and claimRewards as much as they want.

Impact: Any account with a loveToken can claim as much as they want, severely impacting the intended purpose of the protocol.

Proof of Concept: Consider a scenario where two soulmates mint soulmate NFT and attacker joins the protocol.

PoC

```
1
2     function testAnyoneCanDepositAndClaimRewards() public
          soulmatesEntered {
3         address unknown = makeAddr("unknown");
4
5         vm.warp(block.timestamp + 14 days + 1 seconds);
6
7         // Get love token through airdrop
8         vm.prank(unknown);
9         airdropContract.claim();
10
11        uint256 balanceBeforeClaim = loveToken.balanceOf(unknown);
12
13        console2.log("Balance of unknown: %s", balanceBeforeClaim);
14        console2.log("Balance of stakingVault: %s", loveToken.balanceOf
              (address(stakingVault)));
15
16        uint256 balance = 10 ether;
17
18        vm.startPrank(unknown);
19        loveToken.approve(address(stakingContract), balance);
20        stakingContract.deposit(balance);
21
22        stakingContract.claimRewards();
23        vm.stopPrank();
24
25        // Amount claimed through airdrop -> 14 loveTokens
26        // Deposited amount ---------------> 10 loveTokens
27        // Got 20 love tokens using couple ID of 0; userStakes[msg.
              sender] = 10 (deposited) * 2(time passed: 2 weeks) = 20
              loveTokens
28        // Total loveTokens After claiming rewards: 24 loveTokens!
29        uint256 expectedBalance = 14 ether -  10 ether + 20 ether;
30        uint256 balanceAfterClaim = loveToken.balanceOf(unknown);
31        console2.log("Balance of unknown after claim: %s",
              balanceAfterClaim);
32    }
```

The attacker has successfully claimed rewards without minting a single love token!

Recommended Mitigation: Consider minting soulmate NFT starting with nextID = 1. So ID 0 would be returned for addresses that did not mint any Soulmate NFT. Modify the the `Staking::claimRewards()` accordingly to revert when ID 0 is returned.

```
1
```

```
    function claimRewards() public {
        uint256 soulmateId = soulmateContract.ownerToId(msg.sender);
+       if (soulmateId == 0) revert();
        ...
    }
```

[H-03] Incorrect distribution of rewards in `Staking::claimRewards()` due to incorrect tracking/calculation of userStakes and time elapsed.

Description: According to documentation, soulmates can stake collateral and claim rewards, depending on the amount they staked and the time elapsed in weeks. The implementation of calculation is incorrect in `claimRewards()`, where the calculation is based on time elapsed since last claim was made, and if that's 0, then since Soulmate NFT was minted, not when a user's collateral was last deposited.

```
@>  if (lastClaim[msg.sender] == 0) {
@>      lastClaim[msg.sender] = soulmateContract.idToCreationTimestamp(
    soulmateId);
     }
```

Additionally, userStakes is not tracked properly. A user's deposited collateral is not set to zero after a claim has been made. Claiming rewards imply that a user has already used their stakes to get the reward so the staked amount should be set to 0, but it isn't how it is implemented.

Impact:

- Soulmates are rewarded based on when their Soulmate NFT was minted regardless of the time they made deposits.
- Soulmates get rewards based on current deposits as well as previous deposits for which they've already claimed rewards; Users get more

Proof of Concept: Consider the following case:

- Two soulmates join the protocol and Soulmate NFT is minted with Id: 0
- After two weeks, attaker joins and mints loveToken without pairing with someone.
- After getting loveTokens, they stake 10 loveTokens and claim rewards without any time elapse.
- Two weeks after, they deposit 20 loveTokens and get 60 loveTokens as a reward; more than what they should be getting.

PoC

```

function testAnyoneCanDepositAndClaimRewards() public soulmatesEntered
    {
```

```
        address attacker = makeAddr("attacker");

        vm.warp(block.timestamp + 14 days + 1 seconds);

        // Get love token through airdrop
        vm.prank(attacker);
        airdropContract.claim();

        uint256 balanceBeforeClaim = loveToken.balanceOf(attacker);

        console2.log("Balance of attacker: %s", balanceBeforeClaim);
        console2.log("Balance of stakingVault: %s", loveToken.balanceOf
            (address(stakingVault)));

        uint256 balance = 10 ether;

        // No time elapsed since attacker deposited, and they can
            already claimRewards
        vm.startPrank(attacker);
        loveToken.approve(address(stakingContract), balance);
        stakingContract.deposit(balance);

        stakingContract.claimRewards();
        vm.stopPrank();

        // Claimed 14 loveTokens through airdrop
        // Staked 10 love tokens. Remaining: 4 love tokens
        // Got 20 love tokens using couple ID of 0; userStakes[msg.
            sender] = 10 * 2(2 weeks passed) = 20
        // Total loveTokens After claiming rewards: 24!
        uint256 expectedBalance = 14 ether -  10 ether + 20 ether;
        uint256 balanceAfterClaim = loveToken.balanceOf(attacker);
        console2.log("Balance of attacker after claim: %s",
            balanceAfterClaim);

        assertEq(expectedBalance, balanceAfterClaim);
        assertLt(balanceBeforeClaim, balanceAfterClaim);

        // 2 weeks have passed!
        vm.warp(block.timestamp + 14 days + 1 seconds);
        uint256 depositBalance = 20 ether;

        vm.startPrank(attacker);
        loveToken.approve(address(stakingContract), depositBalance);
        stakingContract.deposit(depositBalance);

        stakingContract.claimRewards();
        vm.stopPrank();

        // Total = 24 loveTokens
        // Deposited = 20 loveTokens
```

```
50          // claimReward = 30 (recently deposited 20 + 10 from the last
                deposit) * 2 (2 weeks elapsed) = 60 loveTokens!
51          uint256 expectedBalanceAfter60Days = 24 ether - 20 ether + 60
                ether;
52          uint256 actualBalanceAfter60Days = loveToken.balanceOf(attacker
                );
53          assertEq(expectedBalanceAfter60Days, actualBalanceAfter60Days);
54      }
```

Recommended Mitigation:

- Consider adding lastDeposits mapping reflecting the last time deposits were made, to make sure that a user has deposited and use that to revert in `claimRewards()`. Additionally, reset user deposits to 0 once they claim so they could deposit again to become eligible for another claim.

```
+   error Staking__DepositFirstBeforeClaiming();
+   mapping(address user => uint256 timestamp) public lastDeposit;

    function deposit(uint256 amount) public {
        if (loveToken.balanceOf(address(stakingVault)) == 0) {
            revert Staking__NoMoreRewards();
        }
        // No require needed because of overflow protection
        userStakes[msg.sender] += amount;
+       lastDeposit[msg.sender] = block.timestamp;
        loveToken.transferFrom(msg.sender, address(this), amount);

        emit Deposited(msg.sender, amount);
    }

    function claimRewards() public {
        uint256 soulmateId = soulmateContract.ownerToId(msg.sender);
        // first claim

+       if (lastDeposit[msg.sender] == 0) revert
    Staking__DepositFirstBeforeClaiming();

        if (lastClaim[msg.sender] == 0) {
-           lastClaim[msg.sender] = soulmateContract.
    idToCreationTimestamp(soulmateId);
+           lastClaim[msg.sender] = lastDeposit[msg.sender];
        }

        // How many weeks passed since the last claim.
        // Thanks to round-down division, it will be the lower amount
            possible until a week has completly pass.
        // @audit-high This doesn't track/calculate time elapsed since
            deposit, but time elapsed since lastClaim.
        uint256 timeInWeeksSinceLastClaim = ((block.timestamp -
```

```
            lastClaim[msg.sender]) / 1 weeks);

        if (timeInWeeksSinceLastClaim < 1) {
            revert Staking__StakingPeriodTooShort();
        }

        ...

        lastClaim[msg.sender] = block.timestamp;
        uint256 amountToClaim = userStakes[msg.sender] *
            timeInWeeksSinceLastClaim;
+       userStakes[msg.sender] = 0;
        ...
    }
```

Medium

[M-01] No checks against users that did not mint Soulmate NFt in `Soulmate::writeMessageInSharedSpace()`, so anyone can send messages in a shared space.

Description: The `writeMessageInSharedSpace()` gets ID of a soulmate from `ownerToId` mapping. For someone that is not in the mapping, this returns a default value of 0. Users that are waiting to pair up with someone, or users that haven't minted any Soulmate NFTs can not only send messages in sharedspace of ID 0, but also read its messages.

Impact: Malicious users can send offensive messages resulting in divorce and loss of loveTokens.

Proof of Concept: Add the following test to `SoulmateTest.t.sol`

```

    function testUnknownAddressCanSendMessagesInSpaceUnderIDzero()
        public soulmatesEntered {

        address unknown = makeAddr("unknown");
        string memory message = "Hey! I don't want to be with you! You
            suck!";
        vm.expectEmit();
        emit MessageWrittenInSharedSpace(0, message);

        vm.prank(unknown);
        soulmateContract.writeMessageInSharedSpace(message);

        string memory messageInSpace = soulmateContract.
            readMessageInSharedSpace();
        console2.log("Message: %s", messageInSpace);
    }
```

Recommended Mitigation: Consider pre-incrementing the nextID in `Soulmate::mintSoulmateToken()` so first nextID minted is 1, leaving the ID 0 for non-users.

```
1
2       function mintSoulmateToken() public returns (uint256) {
3           ...
4
5   -       _mint(msg.sender, nextID++);
6   +       _mint(msg.sender, ++nextID);
7
8           ...
9       }
```

Low

[L-01] Misleading Total souls count by `Soulmate::totalSouls()` due to unpaired and self-paired souls

Description: The `totalSouls()` function returns # of active souls in the protocol by doubling the `Soulmate::nextID`. However, it does not account for self-paired souls or souls waiting to get paired, leading to misrepresented count of active souls in the system.

Impact: Inaccurate representation of totalSouls impacts reliability of the protocol's metrics. It could mislead users and stakeholders about platform's activity level, and actual number of successful pairings, potentially affecting user trust and agreement.

Proof of Concept: Add the following test to `SoulmateTest.t.sol`,

PoC

```
1
2       function testIncorrectRepresentationOfTotalSoulsInTheSystem()
           public {
3           vm.prank(soulmate1);
4           soulmateContract.mintSoulmateToken();
5
6           uint256 totalSouls = soulmateContract.totalSouls();
7           assertEq(totalSouls, 0);
8
9           vm.prank(soulmate1);
10          soulmateContract.mintSoulmateToken();
11
12          uint256 totalSoulsCurrently = soulmateContract.totalSouls();
13          assertEq(totalSoulsCurrently, 2);
14      }
```

Recommended Mitigation: The following modifications are recommended.

- Consider adding required/revert statement to `Soulmate::mintSoulmateToken()` to prevent self-pairings.

```
    function mintSoulmateToken() public returns (uint256) {
        // Check if people already have a soulmate, which means already have a token
        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)) {
+           if (soulmate2 == soulmate1) revert();
            idToOwners[nextID][1] = msg.sender;
            // Once 2 soulmates are reunited, the token is minted
            ownerToId[msg.sender] = nextID;
            soulmateOf[msg.sender] = soulmate1;
            soulmateOf[soulmate1] = msg.sender;
            idToCreationTimestamp[nextID] = block.timestamp;

            emit SoulmateAreReunited(soulmate1, soulmate2, nextID);

            // proof-written @audit-info Only one of the soulmates is minted soulbound NFT and
            // it isn't used anywhere in the protocol. SoulboundNFT has no usage in the protocol.
            _mint(msg.sender, nextID++);
        }

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

- Consider changing name of `Soulmate::totalSouls()` to `Soulmate::totalPairs()`.

```
-   function totalSouls() external view returns (uint256) {
+   function totalPairs() external view returns (uint256) {
        return nextID * 2;
    }
```