
BELLUM
GALAXY

Smart Contract Audit Report

General Information

- **Project Name:** First Flight #9: Soulmate
- **Smart Contract Address:** none
- **Audit Date:** 02/10/2024
- **Audit Tools Used:** Code Review, Slither
- **Auditors:** Barba

Executive Summary

A competitive audit by CodeHawks.

Audit Findings

High Vulnerabilities

- **Operational Logic error leading to draining of funds**
    - **Description:**
    - If an user deposit 1 `LoveToken` in the Staking pool in week one without withdrawing, and deposit 10 `LoveToken` in week 6 and then `claimRewards`, the pice of code below will give him the rewards for the total 11 `LoveToken` during the six weeks period and not only 6 `LoveToken` for the first deposit.

```
@>  uint256 timeInWeeksSinceLastClaim = ((block.timestamp -
        lastClaim[msg.sender]) / 1 weeks);

    if (timeInWeeksSinceLastClaim < 1)
        revert Staking__StakingPeriodTooShort();

    lastClaim[msg.sender] = block.timestamp;

    // Send the same amount of LoveToken as the week waited times the
number of token staked
@>  uint256 amountToClaim = userStakes[msg.sender] *
        timeInWeeksSinceLastClaim;
    loveToken.transferFrom(
        address(stakingVault),
        msg.sender,
        amountToClaim
    );
```

    - **Impact:**
    - A malicious user could accumulate a considerable amount of `LoveToken`, deposit and withdraw in a small period of time multiplying the real rewards available for his address.

- **Proof of Code:**

▶ Add the code below into `StakingTest.t.sol`

```
    function test_ClaimRewardsPoC() public {
      uint balance = 11 ether;
      _giveLoveTokenToSoulmates(balance);

      uint balancePerSoulmates = 1 ether;
      uint256 secondBalancePerSoulmates = 10 ether;
      uint weekOfStaking = 5;

      vm.startPrank(soulmate1);
      loveToken.approve(address(stakingContract), balancePerSoulmates);
      stakingContract.deposit(balancePerSoulmates);
      vm.stopPrank();

      vm.warp(block.timestamp + weekOfStaking * 1 weeks + 1 seconds);

      vm.startPrank(soulmate1);
      loveToken.approve(address(stakingContract),
secondBalancePerSoulmates);
      stakingContract.deposit(secondBalancePerSoulmates);
      vm.stopPrank();

      vm.prank(soulmate1);
      stakingContract.claimRewards();

      uint256 rewardsOfSixWeeksAfterDeposit = 66 ether;
      uint256 realBalance = loveToken.balanceOf(soulmate1);

      assertTrue(
          realBalance == rewardsOfSixWeeksAfterDeposit
      );
    }
```

- **Recommendation:**

▶ Recommended measures bellow

```
+   mapping(address owner => uint256 rewards) private s_rewards;

    function deposit(uint256 amount) public {
        if (loveToken.balanceOf(address(stakingVault)) == 0)
            revert Staking__NoMoreRewards();
        // No require needed because of overflow protection

+        s_rewards[msg.sender] = userStakes[msg.sender] *
timeInWeeksSinceLastClaim;
+        lastClaim[msg.sender] =
soulmateContract.idToCreationTimestamp(soulmateId);
```

```
        userStakes[msg.sender] += amount;

        loveToken.transferFrom(msg.sender, address(this), amount);
        emit Deposited(msg.sender, amount);
    }
```

- **Access Control, leading to a loss of funds**
  - **Description:**
    - The `Airdrop::claim` function doesn't have any access control mechanism and this fact allied with a protocol logic allows a malicious actor to steal money from the participants. see the code bellow:

```
    uint256 numberOfDaysInCouple = (block.timestamp -
                soulmateContract.idToCreationTimestamp(
                    soulmateContract.ownerToId(msg.sender)
                )) / daysInSecond;
```

  - **Impact:**
  - Allows malicious actor to steal `LoveToken` from the participants.
  - **Proof of Code:**
    - Soulmate1 and Soulmate2 mint the soulmate NFT
    - They let the Airdrop accumulate during 200 days
    - Then somebody sees and claim with a wallet that doesn't participate in the protocol.
    - This somebody will be consider the owner off token[0] and will be able to claim the rewards.

▶ Add the code bellow on `AirdropTest.t.sol`

```
    address private maliciousActor1 = makeAddr("maliciousActor1");
    address private maliciousActor2 = makeAddr("maliciousActor2");
    function test_ClaimWithoutBeASoulmate() public {
        _mintOneTokenForBothSoulmates();

        // Not enough day in relationship
        vm.prank(soulmate1);
        vm.expectRevert();
        airdropContract.claim();

        vm.warp(block.timestamp + 200 days + 1 seconds);

        vm.prank(maliciousActor1);
        airdropContract.claim();

        assertTrue(loveToken.balanceOf(maliciousActor1) == 200 ether);
```

```
        vm.prank(maliciousActor2);
        airdropContract.claim();

        assertTrue(loveToken.balanceOf(maliciousActor2) == 200 ether);
    }
```

- **Recommendation:**
  - Add a check to limit access to soulmates only

▶ Add the code bellow on `AirdropTest.t.sol`

```
+   error Airdrop__YouMustHaveASoulmateToClaim();
    /// @notice Claim tokens. Every person who have a Soulmate NFT
token can claim 1 LoveToken per day.
    function claim() public {
+       address soulmate = soulmateContract.soulmateOf(msg.sender);
+       if (soulmate == address(0))
+           revert Airdrop__YouMustHaveASoulmateToClaim();

      // No LoveToken for people who don't love their soulmates
anymore.
      if (soulmateContract.isDivorced()) revert
Airdrop__CoupleIsDivorced();

      // Calculating since how long soulmates are reunited
      //@external call before state change
      uint256 numberOfDaysInCouple = (block.timestamp -
          soulmateContract.idToCreationTimestamp(
              soulmateContract.ownerToId(msg.sender)
          )) / daysInSecond;

      uint256 amountAlreadyClaimed = _claimedBy[msg.sender];

      if (
          amountAlreadyClaimed >=
          numberOfDaysInCouple * 10 ** loveToken.decimals()
      ) revert Airdrop__PreviousTokenAlreadyClaimed();

      uint256 tokenAmountToDistribute = (numberOfDaysInCouple *
          10 ** loveToken.decimals()) - amountAlreadyClaimed;

      // Dust collector
      if (
          tokenAmountToDistribute >=
          loveToken.balanceOf(address(airdropVault))
      ) {
          tokenAmountToDistribute = loveToken.balanceOf(
              address(airdropVault)
```

```
            );
        }
        _claimedBy[msg.sender] += tokenAmountToDistribute;

        emit TokenClaimed(msg.sender, tokenAmountToDistribute);

        loveToken.transferFrom(
            address(airdropVault),
            msg.sender,
            tokenAmountToDistribute
        );
    }
```

- **Soulmate being able to withdraw `LoveToken` from the period after divorce, leading to a loss of funds to protocol**
    - **Description:**
        - Even though the parties must accept some kind of loss in a divorce, can not be the protocol. The implemented logic on `Airdrop::claim` allows divorced soulmates to claim tokens after the divorce.
    - **Impact:**
        - Divorced soulmates withdrawing Airdrop funds and breaking the protocol goal.
    - **Proof of Code:**
    - Soulmate1 get divorce from soulmate2
    - Soulmate2 claims the `LoveToken`
    - Soulmate2 awaits 100 days
    - Soulmate2 claims more `LoveToken`

    ▶ Add the code below on `Airdrop.t.sol` file

```
function test_ClaimOfAccumulateLoveTokenAfterDivorce() public {
        _mintOneTokenForBothSoulmates();

        vm.warp(block.timestamp + 200 days + 1 seconds);

        vm.prank(soulmate1);
        soulmateContract.getDivorced();

        vm.prank(soulmate2);
        airdropContract.claim();

        assertTrue(loveToken.balanceOf(soulmate2) == 200 ether);
```

```
        vm.warp(block.timestamp + 100 days + 1 seconds);

        vm.prank(soulmate2);
        airdropContract.claim();

        assertTrue(loveToken.balanceOf(soulmate2) == 300 ether);
    }
```

- **Recommendation:**
  - Adjust the `Soulmate::isDivorced` function as follows:

    ▶ Add the code below on `Soulmate` file

    ```
    -   function isDivorced() public view returns (bool) {
    +   function isDivorced(address _user) public view returns (bool)
    {
    -       return divorced[msg.sender];
    +       return divorced[_user];
        }
    ```

  - Adjust the `Airdrop::claim` function as follows:

    ▶ Add the code below on `Soulmate` file

    ```
      function claim() public {
        address soulmate = soulmateContract.soulmateOf(msg.sender);
        if (soulmate == address(0))
            revert Airdrop__YouMustHaveASoulmateToClaim();

        // No LoveToken for people who don't love their soulmates
    anymore.
    -   if (soulmateContract.isDivorced()) revert
    Airdrop__CoupleIsDivorced();
    +   if (soulmateContract.isDivorced(msg.sender)) revert
    Airdrop__CoupleIsDivorced();

        // Calculating since how long soulmates are reunited
        //@external call before state change
        uint256 numberOfDaysInCouple = (block.timestamp -
            soulmateContract.idToCreationTimestamp(
                soulmateContract.ownerToId(msg.sender)
            )) / daysInSecond;

        uint256 amountAlreadyClaimed = _claimedBy[msg.sender];

        if (
    ```

```
            amountAlreadyClaimed >=
            numberOfDaysInCouple * 10 ** loveToken.decimals()
        ) revert Airdrop__PreviousTokenAlreadyClaimed();

        uint256 tokenAmountToDistribute = (numberOfDaysInCouple *
            10 ** loveToken.decimals()) - amountAlreadyClaimed;

        // Dust collector
        if (
            tokenAmountToDistribute >=
            loveToken.balanceOf(address(airdropVault))
        ) {
            tokenAmountToDistribute = loveToken.balanceOf(
                address(airdropVault)
            );
        }
        _claimedBy[msg.sender] += tokenAmountToDistribute;

        emit TokenClaimed(msg.sender, tokenAmountToDistribute);

        loveToken.transferFrom(
            address(airdropVault),
            msg.sender,
            tokenAmountToDistribute
        );
    }
```

Medium Severity Vulnerabilities

- **Double Call in `Soulmate::mintSoulmateToken`, an user can be his own soulmate breaking the goal of the project**
    - **Description:**
        - An user call double call the `Soulmate::mintSoulmateToken` and be his own soulmate. This way, the protocol purpose is violated.
        - Love yourself is the greatest form of love, however in this scenario is a rule violation too.
    - **Impact:**
        - The project goal, to unite two "unknown soulmates" is violated.
    - **Proof of Code:**
        - User1 Call `Soulmate::mintSoulmateToken` once
        - User1 call `Soulmate::mintSoulmateToken` again
        - User1 mint the token alone, without a soulmate

        ▶ Add the code below in `SoulmateTest.t.sol`

```
     function test_MintNewTokenCalledTwoTimesInARow() public {
         uint tokenIdMinted = 0;

         vm.prank(soulmate1);
         soulmateContract.mintSoulmateToken();

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

         vm.prank(soulmate1);
         soulmateContract.mintSoulmateToken();

         assertTrue(soulmateContract.totalSupply() == 1);
         assertTrue(soulmateContract.soulmateOf(soulmate1) ==
soulmate1);
         assertTrue(soulmateContract.soulmateOf(soulmate1) ==
soulmate1);
         assertTrue(soulmateContract.ownerToId(soulmate1) ==
tokenIdMinted);
     }
```

- **Recommendation:**

▶ Add the code below in `SoulmateTest.t.sol`

```
    function mintSoulmateToken() public returns (uint256) {
+       if(idToOwners[ownerToId[msg.sender]][0] == msg.sender ||
idToOwners[ownerToId[msg.sender]][1] == msg.sender) revert
Soulmate__alreadyIntheSoulmateList();
        // 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)) {
            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);
            _mint(msg.sender, nextID++);
        }
```

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

- **Wrong result in `Soulmate::totalSouls` function, could cause a participant to enter again and be his own soulmate**
    - **Description:**
    - If an USER call the `Soulmate::totalSouls` to see the numbers of participants before apply on `Soulmate::mintSoulmateToken`, and apply being the soulmate[0], when the USER checks `Soulmate::totalSouls` again, the value will be the same. So the user could try `Soulmate::mintSoulmateToken` again and he will became his own soulmate.
    - **Impact:**
    - This problem could cause many people to became their own soulmates.
    - **Proof of Code:**
        1. USER call the `Soulmate::totalSouls`
        2. Receive a number
        3. Call `Soulmate::mintSoulmateToken`
        4. If the USER is the `soulmate[0]`, he will call `Soulmate::totalSouls` and receive the same number
        5. User could call `Soulmate::mintSoulmateToken` again, trying to enter on a second time.
        6. Now the `Soulmate::totalSouls` will update. However, the user will be his own soulmate.

    ▶ Add the code bellow to `Soulmate.t.sol` file

```
function test_MintNewTokenTotalSouls() public {

    vm.prank(soulmate1);
    soulmateContract.mintSoulmateToken();

    vm.prank(soulmate2);
    soulmateContract.mintSoulmateToken();

    assertTrue(totalSouls == 2);

    vm.prank(soulmate3);
    soulmateContract.mintSoulmateToken();

    uint256 totalSouls = soulmateContract.totalSouls();

    assertTrue(totalSouls == 2);
}
```

    - **Recommendation:**

- We can add a counter in `Soulmate::mintSoulmateToken` to count soulmates

  ▶ See the suggestion below

```
+  uint256 private s_soulsCounter;

  function mintSoulmateToken() public returns (uint256) {

      address soulmate = soulmateOf[msg.sender];
      if (soulmate != address(0))
            revert Soulmate__alreadyHaveASoulmate(soulmate);
+       s_soulsCounter++;
        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 totalSouls() external view returns (uint256) {
-   return nextID * 2;
+   return s_soulsCounter;
  }
```

- **Anyone can send a message through the `Soulmate::writeMessageInSharedSpace` so the soulmates with the nft Id0 will have messages sent on his behalf or have override messages**

  - **Description:**

    - Anyone can call the function `Soulmate::writeMessageInSharedSpace` and override the message sent by the soulmate nft id0. Nevertheless, anyone can send an offensive message and cause a divorce leading to a loss of the airdrop rights. Considering the correction of "Soulmate being able to withdraw `LoveToken` from the period after divorce, leading to a loss of funds to protocol" finding, this can even be worse.
    - If an offensive message is sent through the breach, one of the soulmates sees, claims the withdrawal, and immediately gets divorced, the other soulmate will be directly private of

his airdrop funds.

- **Impact:**
  - Malicious users can send an offensive message leading to a divorce and loss of `LoveTokens`.
- **Proof of Code:**

▶ See the code below. PS: This code is already in your test suit

```
  function test_WriteAndReadSharedSpace() public {
    vm.prank(soulmate1);
    soulmateContract.writeMessageInSharedSpace("Buy some eggs");

    vm.prank(soulmate2);
    string memory message =
soulmateContract.readMessageInSharedSpace();

    string[4] memory possibleText = [
        "Buy some eggs, sweetheart",
        "Buy some eggs, darling",
        "Buy some eggs, my dear",
        "Buy some eggs, honey"
    ];
    bool found;
    for (uint i; i < possibleText.length; i++) {
        if (compare(possibleText[i], message)) {
            found = true;
            break;
        }
    }
    console2.log(message);
    assertTrue(found);
    }
```

- **Recommendation:**
- Add a checker in the function. See the example below.

▶ See the code suggestion below.

```
  function writeMessageInSharedSpace(string calldata message) external
{
+     if(soulmateOf[msg.sender] == address(0)) revert
Soulmate__OnlyLoversCanSendMessages();
    uint256 id = ownerToId[msg.sender];
    sharedSpace[id] = message;
    emit MessageWrittenInSharedSpace(id, message);
  }
```

- **Incorrect erc20 interface, a contract compiled with Solidity > 0.4.22 interacting with these functions will fail to execute them, as the return value is missing.**
    - **Description:**
        - Incorrect return values for ERC20 functions. A contract compiled with Solidity > 0.4.22 interacting with these functions will fail to execute them, as the return value is missing.
            - **Instances of incorrect interfaces**
                - `ILoveToken`
                - `ISoulmate`
    - **Impact:**
        - Token.transfer does not return a boolean. Bob deploys the token. Alice creates a contract that interacts with it but assumes a correct ERC20 interface implementation. Alice's contract is unable to interact with Bob's contract.
    - **Proof of Code:**
        - See Slither documentation
    - **Recommendation:**
        - Set the appropriate return values and types for the defined ERC20 functions.

Low Severity Vulnerabilities

- **Vulnerable Randomness Method - Uses a weak PRNG**
    - **Description:**
        - Blockchains are deterministic, so this way of drawing winners is inefficient, insecure and can be manipulated.
        - Using `block.timestamp` creates a predictable final number. A predictable number is not a good random number.
        - Malicious users can manipulate these values or know them ahead of time to choose the complement of the function return.

```
niceWords[block.timestamp % niceWords.length];
```

    - **Impact:**
        - Any user can influence the return of the `Soulmate::readMessageInSharedSpace`.
    - **Proof of Code:**
        - Validators can know ahead of time the `block.timestamp` or `block.difficulty`and use that to predict when/how to participate. See the solidity blog on prevrandao. `block.difficulty` was recently replaced with prevrandao.
        - Using on-chain values as a randomness seed is a well-documented attack vector in the blockchain space.
    - **Recommendation:**

- Considering that te application of the vulnerable code don't result in any loss and would be pretty expensive to manipulate a function like that on Ethereum, there is no need to change the approach. However, this is not a best practice. Consider using a cryptographically provable random number generator such as Chainlink VRF in this cases.

Informational Observations

- **Empty function, can add vulnerabilities after the review**
  - **Description:**
  - The function `Soulmate::tokenURI` is empty and documentation let us believe that will be completed after the review.The function could introduce some vulnerabilities in the contract.

    ▶ See the code below

    ```
      function tokenURI(uint256) public pure override returns (string
    memory) {
          // To do
          return "";
      }
    ```

  - **Impact:**
  - Unknown
  - **Recommendation:**
  - Ask for a new review after complete the code.
- **Floating Pragma**
  - **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 unintended results.
  - https://swcregistry.io/docs/SWC-103/

    ```
        pragma solidity ^0.8.23;
    ```

  - **Recommendation:**

    ```
    -   pragma solidity ^0.8.23;
    +   pragma solidity 0.8.23;
    ```

- **Message between soulmates are public**

- **Description**
  - The mapping that stores the messages sent by soulmates is public. So anyone can easily see the messages.
- **Recommendation**

```
- mapping(uint256 id => string) public sharedSpace;
+ mapping(uint256 id => string) private sharedSpace;
```

- **CEI Patterns not followed**
  - **Description**
    - The following functions doesn't follow CEI patterns. It's better to follow CEI (Checks, Effects, Interactions).

▶ See the code and adjustment suggestions bellow

```
    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;

+       emit Deposited(msg.sender, amount);
        loveToken.transferFrom(msg.sender, address(this), amount);
-       emit Deposited(msg.sender, amount);
    }
```

```
    function withdraw(uint256 amount) public {

        // No require needed because of overflow protection
        userStakes[msg.sender] -= amount;

+       emit Withdrew(msg.sender, amount);
+       loveToken.transfer(msg.sender, amount);
-       emit Withdrew(msg.sender, amount);
    }
```

```
    function claimRewards() public {
        uint256 soulmateId =
soulmateContract.ownerToId(msg.sender);
        // first claim
        if (lastClaim[msg.sender] == 0) {
            lastClaim[msg.sender] =
```

```
soulmateContract.idToCreationTimestamp(
                soulmateId
            );
        }

        // How many weeks passed since the last claim.
        // Thanks to round-down division, it will be the lower
amount possible until a week has completely pass.
        uint256 timeInWeeksSinceLastClaim = ((block.timestamp -
            lastClaim[msg.sender]) / 1 weeks);

        if (timeInWeeksSinceLastClaim < 1)
            revert Staking__StakingPeriodTooShort();

        lastClaim[msg.sender] = block.timestamp;
        // Send the same amount of LoveToken as the week waited
times the number of token staked
        uint256 amountToClaim = userStakes[msg.sender] *
            timeInWeeksSinceLastClaim;

+       emit RewardsClaimed(msg.sender, amountToClaim);
        loveToken.transferFrom(
            address(stakingVault),
            msg.sender,
            amountToClaim
        );
-       emit RewardsClaimed(msg.sender, amountToClaim);
    }
```

```
    function initVault(ILoveToken loveToken, address
managerContract) public {
        if (vaultInitialize) revert Vault__AlreadyInitialized();
+       vaultInitialize = true;
        loveToken.initVault(managerContract);
-       vaultInitialize = true;
    }
```

- **Recommendation**
  - Use CEI. See recommendation above

Gas Observations

- **Gas cost to send text messages on Ethereum**
  - **Description:**
  - The `Soulmate::writeMessageInSharedSpace` function implements a logic which users send text messages on-chain, however the gas costs on the Ethereum blockchain will turn this practice too expensive.
  - **Recommendation:**

    - Use the wallet address to sign the messages through the frontend application.

- **Unchanged state variables should be declared constant or immutable**

    - **Description**
        - Reading from storage is much more expensive than reading from a constant or immutable variable.
        - **Instances**

            - Constant Instances:

```
  PuppyRaffle.commonImageUri (src/PuppyRaffle.sol#35) should
be constant
  PuppyRaffle.legendaryImageUri (src/PuppyRaffle.sol#45)
should be constant
  PuppyRaffle.rareImageUri (src/PuppyRaffle.sol#40) should be
constant
```

            - Immutable Instances:

```
  PuppyRaffle.raffleDuration (src/PuppyRaffle.sol#21) should
be immutable
```

- **Internally unassessed functions should be declared external**

    - **Description**
        - Public function is much more expensive than External function.
        - **Instances**

            - Found in src/Airdrop.sol Line: 51

```
    function claim() public {
```

            - Found in src/LoveToken.sol Line: 48

```
    function initVault(address managerContract) public {
```

            - Found in src/Soulmate.sol Line: 63

```
    function mintSoulmateToken() public returns (uint256) {
```

            - Found in src/Soulmate.sol Line: 93

```
    function tokenURI(uint256) public pure override returns
(string memory) {
```

- Found in src/Soulmate.sol Line: 100

```
    function transferFrom(address, address, uint256) public
pure override {
```

- Found in src/Soulmate.sol Line: 128

```
    function getDivorced() public {
```

- Found in src/Soulmate.sol Line: 135

```
    function isDivorced() public view returns (bool) {
```

- Found in src/Staking.sol Line: 51

```
    function deposit(uint256 amount) public {
```

- Found in src/Staking.sol Line: 65

```
    function withdraw(uint256 amount) public {
```

- Found in src/Staking.sol Line: 77

```
    function claimRewards() public {
```

- Found in src/Vault.sol Line: 28

```
    function initVault(ILoveToken loveToken, address
managerContract) public {
```

Conclusion

This Security Review had a span of 12 hours duration. Besides the small period, I have found vulnerabilities that could corrupt the protocol and cause funds loss to the protocol and its users. The review purpose was to get experience in this area and this is my first audit report for a competition. To fulfill the competition parameters, some of the findings are not listed here because I wasn't able to prove them. However, this was a great thermometer to find my weakness and work on top of it.
