SOULMATE AUDIT REPORT

Version 0.1

Daniel Vigo

February 14, 2024

SOULMATE AUDIT REPORT

Daniel Vigo

February 14, 2024

Soulmate Audit Report

Prepared by: Daniel Vigo Lead Auditors:

- Daniel Vigo

Assisting Auditors:

- None

Table of Contents

See table

Disclaimer

Daniel Vigo team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the 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.

Audit Details

The findings described in this document correspond to the main commit hash —> Branch main / master

Protocol Summary

Soulmate is a protocol where people can have others people as soulmates and every couple can stake their earned NFT's and claim extra rewards in LoveTokens. The protocol is designed to be used by multiples users, and is not designed to be used by a single user.

Roles

- Owner: Only deploy contracts, no one functionality is restricted to the owner.

- Users: They find a soulmate and earn LoveTokens

Executive Summary

Issues Found

Severity	Number of issues Found
High	3
Medium	4
Low	10
Info	14
Gas Optimizations	0
Total	31

Findings

Soulmate.sol

High

[H1] In the the function ‘mintSoulmateToken()’ only one person can mint the NFT.

Description: In this function, both the people should be able to mint the nft, but only the person who join as second in the mapping can actually mint it.

Impact: An user can mint the ERC721, but his soulmate not.

Proof of Concept: Run this test:

```
function test__Soulmate__OnlyOnePeopleCanMintTheNft()public{
address soulmate1 = address(1);
address soulmate2 = address(2);
vm.startPrank(soulmate1);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate2);
soulmate.mintSoulmateToken();
vm.stopPrank();
uint256 soulmate2Balance = soulmate.balanceOf(soulmate2);
uint256 soulmate1Balance = soulmate.balanceOf(soulmate1);
assertNotEq(soulmate2Balance, soulmate1Balance);
}
```

Recommended Mitigation: Update the logic of the contract, maybe assign with a mapping the possibility to mint the ERC721 token also for the first soulmate user.

Soulmate.sol

[H2] The mapping ‘soulmateOf’ is not well implemented and updated

Description: This mapping is not updated, especially in ‘getDivorced()’, this can potentially cause a big issue in the protocol.

Impact: Others contract can’t actually see if msg.sender has actually a soulmate or not if this value is not updated, also, after user calls the function ‘getDivorced’ he will never have more the possibility to have a new soulmate because ‘soulmateOf’ has not been updated.

Proof of Concept: Run this Test:

```
function test__Soulmate__SoulmateOfNotWellImplemented()public{
address soulmate1 = address(1);
address soulmate2 = address(2);
vm.startPrank(soulmate1);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate2);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate1);
soulmate.getDivorced();
assertEq(soulmate.isDivorced(), true);
address checkSoulmate = soulmate.soulmateOf(address(1));
assertEq(checkSoulmate, soulmate2);
}
```

Recommended Mitigation: Update this mapping every time is needed, so for example also in ‘getDivorced’.

Soulmate.sol

Medium

[M-1] People can have themself as soulmates and mint multiples tokens

Description: In the function ‘mintSoulmateToken’, people have the possibility to mint a large amount of tokens because they can have themself as soulmates, and so they can create multiples accounts and mint a lot of tokens.

Impact: Users (no need to be hacker) can mint more tokens.

```
function test__Soulmate__AnUserCanMintATokenWithHimSelf()public{
address soulmate1 = address(1);
address soulmate2 = address(2);
vm.startPrank(soulmate1);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate1);
soulmate.mintSoulmateToken();
vm.stopPrank();
}
```

Proof of Concept: Run this test and you will see this will not revert:

Recommended Mitigation: It is recommended to require that soulmates can’t be the same address.

Low

[L-1] The mapping ‘ownerToId’ is updated with a value different than that for mint.

Description: In the function ‘mintSoulmateToken’, this mapping is updated with ‘nextID’, but in the mint function the id minted is ‘nextID++’.

Impact: The mapping ‘ownerToId’ and the token minted do not correspond.

Recommended Mitigation: Simply update those 2 values with the exact same number.

[L-2] Different message In shared space from mapping to function.

Description: In the function ‘readMessageInSharedPlace’, people can read the message with also the ‘nice words’ at the end, but this does not happen also if interacting directly with the mapping.

Impact: Different messages from mapping to function

Proof of Concept: Run this test:

```
function test__Soulmate__DifferentsMessagesFromMappingToFunction()public{
address soulmate1 = address(1);
address soulmate2 = address(this);
vm.startPrank(soulmate1);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate2);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate1);
soulmate.writeMessageInSharedSpace("Hey, how are you?");
string memory messageReadFromFunc =
soulmate.readMessageInSharedSpace();
string memory messageReadFromMapping = soulmate.sharedSpace(0);
assertNotEq(messageReadFromFunc, messageReadFromMapping);
}
```

[L-3] Randomness in ‘readMessageInSharedSpace()’.

Description: In this function, it returns at the end of the message a random word in ‘niceWords’ array, but it not well implemented, it returns a word doing ‘block.timestamp % nicewords.length”. This is a bad practice because block.timestamp can be manipulated from the attacker. Also, this practice will ever returns the word ‘darling’, and not the others ones.

Impact: The message is always read with ‘darling’ as last word.

Proof of concept: Run this test

```
function test__Soulmate__RandomnessInReadMessageSharedSpace()public{
address soulmate1 = address(1);
address soulmate2 = address(this);
vm.startPrank(soulmate1);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate2);
soulmate.mintSoulmateToken();
vm.stopPrank();
vm.startPrank(soulmate1);
soulmate.writeMessageInSharedSpace("Hey, how are you?");
string memory messageReadFromFunc =
soulmate.readMessageInSharedSpace();
string memory messageReadFromMapping = soulmate.sharedSpace(0);
assertEq(messageReadFromFunc,"Hey, how are you?, darling");
}
```

Recommended Mitigation: Do not utilize block.timestamp, in order to get random values consider using chainlink VRFA.

Soulmate.sol

Informational

[I-1] Multiples solidity versions (bad)

Description: It is recommended to use a single solidity version and not multiples like ‘^0.8.18’.

Recommended Mitigation: Consider using a single solidity version like ‘0.8.18’.

[I-2] Solidity version too recent

Description: It is recommended to use a solidity version more recent, “0.8.23” is pretty young and it could have vulnerabilities.

Recommended Mitigation: Consider using ‘0.8.18’

[I-3] Messages visible by everyone

Description: In this contract, all messages sent by an user to his soulmate are visible by everyone for two reasons: 1) These values are on blockchain, 2) Mapping are also public

Recommended mitigation: It is impossible to do not let others see a message written on the blockchain (evm), but at least consider make internal or private the mapping ‘sharedSpace’

ISoulmate.sol

Informational

[I-1] Interface of transferFrom bad implementation

Description: In this contract, the function transferFrom is bad implemented, because it does not returns the bool value. This one is informational because the function ‘transferFrom’ can’t be called with success and would revert at the same way.

Recommended Mitigation: Fix the bad line of code with this:

```
function transferFrom(address from, address to, uint256 id) external
returns(bool);
```

LoveToken.sol

Informational

[I-1] ‘import {ISoulmate}’ is not utilized in the contract

Description: this solidity file has been imported into the contract but it is not utilized.

Recommended Mitigation: Do not import this file if it is not needed.

[I-2] Value ‘soulmateContract’ is never used.

Description: In this contract, this value is never used and it is worthless.

Recommended Mitigation: Do not use this value and cancel it from the file if it is not needed.

[I-3] Miss of a check 0 address in ‘initVault’

Description: In the function ‘initVault’ there is a miss of check 0 address on the argument ‘managerContact’

Recommended Mitigation: Add a line of code at the start of the function In order to get sure this is not a 0 address:

```
require(managerContract != address(0), “0 address detected”);
```

ILoveToken.sol

Medium

[M-1] Bad interface implementation of approve()

Description: This contract supports a bad interface implementation of approve because it does not returns a bool value.

Impact: Interacting with this function can cause arithmetics problems.

Recommended Mitigation: Use instead this line of code:

```
function approve(address to, uint256 amount) external returns(bool);
```

ILoveToken.sol

[M-2] Bad interface implementation of transfer()

Description: This contract supports a bad interface implementation of transfer because it does not returns a bool value.

Impact: Interacting with this function can cause arithmetics problems.

Recommended Mitigation: Use instead this line of code:

```
function transfer(address to, uint256 amount) external returns(bool);
```

[M-3] Bad interface implementation of transferFrom()

Description: This contract supports a bad interface implementation of transferFrom because it does not returns a bool value.

Impact: Interacting with this function can cause arithmetics problems.

Recommended Mitigation: Use instead this line of code:

```
function transferFrom(address from, address to, uint256 amount) external
returns(bool);
```

Vault.sol

Low

[L-1] CEI not followed in ‘initVault()’

Description: Possible reentrancy in this function because CEI is not followed, a value is modified before the external call.

Recommended Mitigation: Fix this function in this one:

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

Informational

[I-1] Miss permission check in ‘initVault’

Description: In this confidential function is better let only the owner have access to this one.

Recommended Mitigation: Consider allow only the owner to have access to this function.

[I-3] Miss of a check 0 address in ‘initVault’

Description: In the function ‘initVault’ there is a miss of check 0 address on the argument ‘managerContact’

Recommended Mitigation: Add a line of code at the start of the function In order to get sure this is not a 0 address:

Airdrop.sol

High

[H-1] Everyone can claim 1 token day also if users are not in a couple

Description: In this function, everyone can claim tokens because the 'soulmateOf' of the sender is not well checked.

Impact: Users without a NFT and not in a couple can mint tokens.

Proof of Concept: Run this test:

```
function test__AirdropEveryoneCanClaim1TokenPerDay()public{
vm.startPrank(address(1));
vm.warp(block.timestamp + 87400); //--> A few more than 1 day
airdropContract.claim();
assertEq(loveToken.balanceOf(address(1)), 1 * 1e18);
}
```

Recommended Mitigation: Make sure that 'msg.sender' actually has a soulmate checking for 'Soulmate__soulmateOf(msg.sender)'.

Airdrop.sol

Informational

[I-1] Possible underflow / overflow in claim() 1

Description: There is a possible overflow / underflow in Airdrop__claim__'numberOfDaysInCouple * 10 ** love token.decimals()' due to a multiplication on a result of a division.

Impact: None in this case

Recommended Mitigation: Consider adding an if statement or a 'require' in order to see In a anticipate way if 'numberOfDaysInCouple' returns a 0.

[I-2] Possible underflow / overflow in claim() 2

Description: There is a possible overflow / underflow in Airdrop__claim__'(numberOfDaysInCouple * 10 ** love token.decimals()) - amountAlreadyClaimed' due to a multiplication on a result of a division.

Impact: None in this case

Recommended Mitigation: Consider adding an if statement or a 'require' in order to see In a anticipate way if 'numberOfDaysInCouple' returns a 0.

Staking.sol

Low

[L-1] Bad balance check in deposit()

Description: In the function deposit() the line of code ‘if(loveToken.balanceOf(address(stakingVault)) = 0 ‘ makes not that much sense.

Impact: This has not that much sense because imagine a situation when stakingVault balance is not 0 but is lower than the amount deposited, this is also obviously a possible bug.

Recommended Mitigation: Check if stakingVault balance has that ‘amount’ of tokens deposited * 2 :

```
require(loveToken.balanceOf(address(stakingVault) >= amount * 2), ‘Staking
contract has not enough tokens”);
```

[L-2] CEI not followed In deposit()

Description: CEI is not followed in deposit() because the Emit should be placed before the ‘transferFrom’

Impact: Possible Reentrancy

Recommended Mitigation: Place the emit before transferFrom()

Staking.sol

[L-3] CEI not followed in withdraw()

Description: CEI is not followed in withdraw() because the Emit should be placed before the 'transfer'

Recommended Mitigation: Place the emit before transfer()

[L-4] CEI not followed in claimRewards()

Description: CEI is not followed in claimRewards() because the Emit should be placed before the 'transferFrom'

Impact: Possible Reentrancy

Recommended Mitigation: Place the emit before transferFrom()

[L-5] Underflow / Overflow in claimRewards()

Description: In 'Staking__claimRewards__userStakes[msg.sender] * timeInWeeksSinceLastClaim' there is a multiplication on the result of a division.

Impact: Possible Underflow / Overflow that in this case is not a big problem

Recommended Mitigation: Check if 'timeInWeeksSinceLastClaim' is greater than 0 before the multiplication.

Staking.sol

[L-6] soulmateId bad check in claimRewards()

Description: In this function if msg.sender never interacted with soulmate.sol the value ‘soulmateId[msg.sender] would returns 0 and this can be not that correct.

Impact: All people can call this function (but not receive tokens if they did not staked their tokens).

Recommended Mitigation: Consider check if msg.sender has actually a soulmate and if he staked the tokens.

Informational

[I-1] Bad permissions check in the withdraw function

Description: In this function it is not checked if msg.sender can withdraw that exact amount of tokens.

Impact: Possible Arithmetic error

Recommended Mitigation: Consider check if msg.sender has staked that ‘amount’ of tokens and if he can withdraw them.

[I-2] Bad permissions check in the deposit function

Description: In this function it is not checked if msg.sender can deposit that exact amount of tokens and of course, if he has them.

Impact: Possible Arithmetic error

Recommended Mitigation: Consider check if msg.sender has that ‘amount’ of tokens and if he can deposit them.

Staking.sol

[I-3] Worthless assignment in claimRewards

Description: In this function the ‘lastClaim[msg.sender] = soulmateContract.idToCreationTimestamp(soulmateId)’ is completely worthless because in the next line son code there is ‘lastClaim[msg.sender] = block.timestamp’, so for this reason the first assignment is worthless.

Recommended Mitigation: Consider delete the first assignment.