QuillAudits

Audit Report
March, 2022

For

TheLoveChain

is forever

Contents

Overview

Lovechain

Scope of the Audit

The scope of this audit was to analyze Lovechain smart contract's codebase for quality, security, and correctness.

Lovechain Codebase:
https://github.com/Nextgeniusentrepreneurs/lovechain

Commit hash: a568a3435b248d0b43b77ea4ae364d19c3d6666ct
Contracts:

- LoveGovernanceToken
- LoveToken
- MasterChef
- RevenuePool
- Timelock
- TokenTimelock

Fixed In: c56c7f2c9731fed44672b0fb56b77fe89328dad2

Branch: V3

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Slither, Solhint.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	3	2
Closed	1	1	5	5

Issues Found – Code Review / Manual Testing

High severity issues

1. **Token transfers and mints does not facilitate movement/transfer of delegate votes**

   Function overrides the minting and burning of tokens by facilitating the movement/transfer of delegate votes along with them.

```
function mint(address _to, uint256 _amount) public onlyOwner {
    _mint(_to, _amount);
    _moveDelegates(address(0), _delegates[_to], _amount);
}

function burn(address _from, uint256 _amount) public onlyOwner {
    _burn(_from, _amount);
    _moveDelegates(_delegates[_from], address(0), _amount);
}
```

However, as the token inherits from BEP20 standard. It comes with vanilla transfer and owner-only mint functionality,

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal {
    require(sender != address(0), 'BEP20: transfer from the zero address');
    require(recipient != address(0), 'BEP20: transfer to the zero address');

    _balances[sender] = _balances[sender].sub(amount, 'BEP20: transfer amount exceeds balance');
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}
```

Ref: BEP20 _transfer function

```
function mint(uint256 amount) public onlyOwner returns (bool) {
    _mint(_msgSender(), amount);
    return true;
}
```

Ref: BEP20 mint onlyOwner function

which does not consider/facilitate delegate vote transfers. As a result, users can transfer their tokens without losing/transfering delegate votes to target’s delegate

Recommendation: Consider reviewing and verifying the business and operational logic and modifying/overriding the transfer logic to facilitate movement/transfer of delegate rights. Same should be considered for the owner-only mint function.

V2: Update: The contract has now been updated, by adding an access control for the transfer and transferFrom functions, and these now will be governed by a role as POOL_ROLE, also the owner-only mint functionality has been removed from the imported BEP20. However, the issue remains the same. The transfer functions do not consider/facilitate delegate vote transfers with them, and thus are subject to be fixed.

V3: Update: The contract has now been further updated with the following changes

```
28      function transfer(address _recipient, uint256 _amount) public override onlyRole(POOL_ROLE) returns (bool) {
29          bool state =  super.transfer(_recipient, _amount);
30          if(state && _delegates[msg.sender] != address(0)){
31              if(_delegates[_recipient] == address(0)){
32                  _delegates[_recipient] = _recipient;
33              }
34            _moveDelegates(_delegates[msg.sender], _recipient, _amount);
35          }
36          return state;
37      }
38
39      function transferFrom(
40          address sender,
41          address recipient,
42          uint256 amount
43      ) public override onlyRole(POOL_ROLE) returns (bool) {
44          bool state =  super.transferFrom(sender, recipient, amount);
45          if(state && _delegates[sender] != address(0)){
46              if(_delegates[recipient] == address(0)){
47                  _delegates[recipient] = recipient;
48              }
49            _moveDelegates(_delegates[sender], recipient, amount);
50          }
51          return state;
52      }
```

However, the changes implemented don’t fix the bug.
Exploit Scenario 1:

Let’s assume,
Address X (having POOL_ROLE), doesn’t have any delegate.
Address Y has a delegate as B.

Any transfers made by address X to address Y, will not facilitate movement of delegate rights, and thus address B will not get any voting

rights. Now, if address Y tries to delegate to another address, let's say address C. The operation will fail, as _moveDelegates will first try to remove the voting rights from the current delegate, and as the current delegate doesn't have any voting rights, the operation will always fail.

Exploit Scenario 2:
Let's consider the same scenario explained above, but this time, address X does have a delegate.

Now, the token transfers made by address X to address Y, will move delegate rights from address X's delegate to Y, instead of Y's(recipient's) delegate . Now if Y tries to delegate to another address, it will still not be able to, due to the fact that the voting rights were delegated to its own address and not it's delegate, and for the same reason _moveDelegates will fail as the current delegate doesn't have any voting rights to be removed and transferred to the new desired address.

Recommendation: Consider reviewing and verifying the business and operational logic, and consider transferring the delegate rights from sender's delegate to recipient's delegate.

V4: Update: The contract has now been further updated with the following changes

```
28      function transfer(address _recipient, uint256 _amount) public override onlyRole(POOL_ROLE) returns (bool) {
29          bool state =  super.transfer(_recipient, _amount);
30          if(state){
31              if(_delegates[_recipient] == address(0)){
32                  _delegates[_recipient] = _recipient;
33              }
34            _moveDelegates(_delegates[msg.sender], _delegates[_recipient], _amount);
35          }
36          return state;
37      }
38
39      function transferFrom(
40          address sender,
41          address recipient,
42          uint256 amount
43      ) public override onlyRole(POOL_ROLE) returns (bool) {
44          bool state =  super.transferFrom(sender, recipient, amount);
45          if(state){
46              if(_delegates[recipient] == address(0)){
47                  _delegates[recipient] = recipient;
48              }
49            _moveDelegates(_delegates[sender], _delegates[recipient], amount);
50          }
51          return state;
52      }
```

Exploit Scenario:
Let's say, address A doesn't have any delegate.
MASTER_CHEF_ROLE mints 1000 tokens to address A. Address A now has 1000 tokens, but since there was no delegate, A only has 1000 tokens and no voting rights for any delegate.

Now, POOL_ROLE transfers 500 tokens to address A. Now as address A doesn't have any delegate, the function will make address A as a delegate for itself(by just changing the address in _delegates mapping) and delegate the voting rights. Now A has 1500 tokens and 500 voting rights to itself.

Now if A, wants to delegate to an address B, it will never be possible, because the function will try to subtract 1500 voting rights from existing delegate, but address A has only 500 voting rights, and thus will result in subtraction error, and will revert.

Status: Fixed

Medium severity issues

RevenuePool

2. Missing important checks for contract initialization

Function initialize as the name suggests is intended to initialize the contract. However, it lacks important checks, hence is prone to incorrect contract initialization.

Some of the important checks are:
- Zero address check for stakedToken and admin addresses
- Value checks for reward amount and period end time for the first staking period

Recommendation: Consider adding required checks to avoid risks of incorrect contract initialization.

Status: Fixed

Low severity issues

3. Old Solidity Compiler

Contracts uses an old solidity compiler as 0.6.12, which contains some unpatched bugs.

Recommendation: Use the latest compiler version in order to avoid bugs introduced in older versions.

Status: Fixed

4. Multiple pragma directives have been used

Recommendation: Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the pragma (for e.g. by not using ^ in pragma solidity 0.5.10) ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs. Ref: Security Pitfall 2

Status: Acknowledged

5. Missing two step process to change privileged role Owner

When privileged roles are being changed, it is recommended to follow a two-step approach: 1) The current privileged role proposes a new address for the change 2) The newly proposed address then claims the privileged role in a separate transaction. This two-step change allows accidental proposals to be corrected instead of leaving the system operationally with no/malicious privileged role. (Ref: Security Pitfalls: 162)

However, token contract uses Ownable contract to define a privileged role owner, which uses a one step process to transfer/renounce ownership, as a consequence the owner may accidently lose control over the operational logic of the contract.

Recommendation: Consider switching to a two-step process for transferring ownership and an optional time-margin for renouncing ownership

Status: Fixed
"Ownable contract has been modified to add extra parameter as confirmationToken for renouncing/transferring ownership, which will act as an additional safety check. However, the process is still somewhat single step."

6. Lack of Proper Documentation

Documentation describes what (and how) the implementation of different components of the system does to achieve the specification goals. Without documentation, a system implementation cannot be evaluated against the specification for correctness and one will have to rely on analyzing the implementation itself. (Ref. Security Pitfalls: 137)

However, Operational logic for many functions was not well documented.

Status: Acknowledged

7. **block.timestamp dependance**

Contract uses block.timestamp to calculate staking period, which may be manipulated by miners in order to bypass checks implemented by the contract.

Recommendation: Avoid using block.timestamp for calculating time values.

Status: Acknowledged

8. **Use of token contracts instead of interfaces**

Contracts deal with BEP20 tokens namely LoveToken and LoveGovernanceToken. However, instead of using token interfaces, some contracts try to import the complete token contracts which come with already imported libraries and contracts. Hence, may result in compilation issues.

Some instances of such imports are:
LoveGovernanceToken imports LoveToken.
MasterChef imports both LoveToken and LoveGovernanceToken.

Recommendation: Use token interfaces instead of token contracts to avoid compilation issues.

Status: Fixed

9. Missing Zero Address Checks

Contracts lack zero address checks, hence are prone to be initialized with zero addresses.

LoveGovernanceToken: Constructor lacks zero address check for LoveToken.

MasterChef: Constructor lacks zero address checks for LoveToken and LoveGovernanceToken addresses.

TokenTimelock: Constructor lacks zero address checks for token and beneficiary addresses.

Recommendation: Consider adding zero address checks in order to avoid risks of incorrect contract initializations.

Status: Fixed
“Initialization of LoveToken has been removed from LoveGovernanceToken’s constructor in the latest release”

RevenuePool

10. Unsafe Arithmetic Calculation

Although, contract uses SafeMath library which defines functions to perform safe arithmetic calculations, still some instances of unsafe arithmetic calculations have been reported which are prone to integer overflows. However the likelihood of such an event to occur is low.

Some of the instances found during the audit are:
[#L103-106] require(stakingInfo[nStakingPeriods].periodEndTime - 75 days > block.timestamp, "Staking deposit delay expired");

[#L242-286] pending += user.amount.mul(stakingInfo[lastUpdate].accTokenPerShare).div(PRECISION_FACTOR);

Recommendation: Consider using SafeMath to avoid risks of unsafe arithmetic calculations like Integer Overflows

Status: Fixed

Informational issues

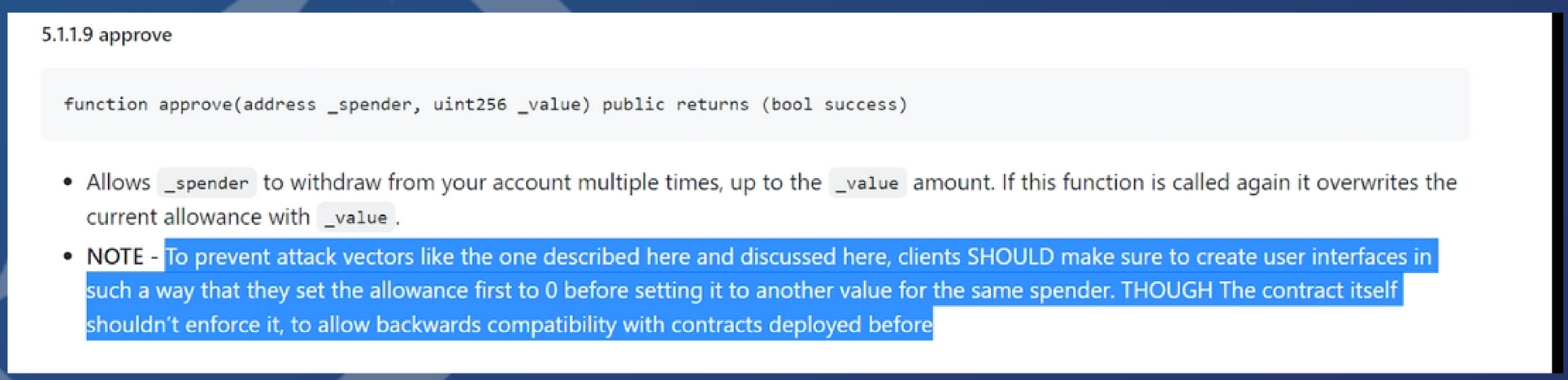
11. **Public functions never used by the contract internally should be declared external to save gas**

    Status: Acknowledged

12. **BEP20 approve race condition**

    The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.

    As the BEP20 standard is proposed by deriving the ERC20 protocol of Ethereum, the race condition exists here as well.


5.1.1.9 approve

```
function approve(address _spender, uint256 _value) public returns (bool success)
```

- Allows `_spender` to withdraw from your account multiple times, up to the `_value` amount. If this function is called again it overwrites the current allowance with `_value`.
- NOTE - To prevent attack vectors like the one described here and discussed here, clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. THOUGH The contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before

Reference:

1. https://eips.ethereum.org/EIPS/eip-20
2. https://github.com/binance-chain/BEPs/blob/master/BEP20.md#5119-approve

Status: Acknowledged

LoveToken

13. Unused internal Function

Contract contains the function getChainId to calculate the chain id of the current chain. However, the function doesn't take part in the operational logic of the contract.

```
function getChainId() internal pure returns (uint256) {
    uint256 chainId;
    assembly {
        chainId := chainid()
    }
    return chainId;
}
```

Recommendation: Consider reviewing and verifying the business and operational logic. Function may be removed in order to optimize the bytecode and save deployment cost.

Status: Fixed

14. The updated contract now introduces some compilation errors

```
pragma solidity >=0.8.4;

import "./BEP20.sol";
import "./interfaces/ILoveToken.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

// LovToken with Governance.
contract LoveToken is ILoveToken, Ownable, BEP20("Lovechain Token", "LOV") {
    /// @notice Creates `_amount` token to `_to`.
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
    }
}
```

As, the token now inherits from ILoveToken token interface. There is a need to use the override keyword for the mint function. However, there is no need for the token to inherit from its own interface here.

Recommendation: Consider removing the inheritance of ILoveToken interface or using the override keyword for overriding the mint function.

Status: Fixed

LoveGovernanceToken

15. Contract defines safeLovTransfer as an owner-only function in order to allow the owner to withdraw Love tokens from the contract. However, it is supposed to transfer ownership to MasterChef contract in order to allow users to stake their Love Tokens in order to get an equal amount of Love Governance tokens.

    Hence, transferring ownership may lead to unoperational safeLovTransfer.

    Recommendation: Consider reviewing and verifying the business and operational logic.

    Status: Fixed

16. **The updated contract now introduces some compilation errors**

```
pragma solidity >=0.8.4;

import "./LoveToken.sol";
import "./interfaces/ILoveGovernanceToken.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

// LoveGovernanceToken with Governance.
contract LoveGovernanceToken is ILoveGovernanceToken, BEP20("Love Governance Token", "gLOV"), AccessControl {
    bytes32 public constant MASTER_CHEF_ROLE = keccak256("MASTER_CHEF_ROLE");
    bytes32 public constant POOL_ROLE = keccak256("POOL_ROLE");
```

    As, the token now inherits from ILoveGovernanceToken token interface. There is a need to use the override keyword for the functions which are being overridden. However, there is no need for the token to inherit from its own interface here.

    Recommendation: Consider removing the inheritance of ILoveGovernanceToken interface or using the override keyword for functions being overriden.

    Status: Fixed

RevenuePool

17. accTokenPerShare of a staking period may provide incorrect information

accTokenPerShare holds the token per share rate, and is updated for every deposit and withdrawal as:

```
266       function _updateTokenShare() internal {
267           uint256 stakedTokenSupply = stakedToken.balanceOf(address(this));
268
269           if (stakedTokenSupply == 0) {
270               lastPoolUpdate = block.timestamp;
271               return;
272           }
273           stakingInfo[nStakingPeriods].accTokenPerShare = stakingInfo[nStakingPeriods]
274               .rewardAmount
275               .mul(PRECISION_FACTOR)
276               .div(stakedTokenSupply);
277       }
```

However, in certain cases, it can provide incorrect information to end users.

Example Scenario:

Let’s assume for staking period 1, stakedTokenSupply is 500, staked by userA(100 token), userB and userC(200 each), rewardAmount is also 500. The calculated accTokenPerShare will be 1.

Now, consider withdrawals
A withdraws all the balance, accTokenPerShare = 1.25
B withdraws all the balance, accTokenPerShare = 2.5

Now, C also withdraws all the balance, here accTokenPerShare will hold the old value of 2.5 because the function logic doesn’t account for these scenarios.

As a result, end users may be receiving incorrect information for accTokenPerShare

Recommendation: Consider reviewing and verifying the business and operational logic

Status: Fixed

Functional Testing Results

LoveToken

- [x] owner should be able to mint tokens
- [x] revert minting in case of unauthorized user

LoveGovernanceToken

- [x] owner should be able to mint/burn governance tokens
- [x] Minting and Burning should also move/transfer delegate voting rights
- [x] Token transfers should also move/transfer delegate voting rights
- [x] Owner should be able to withdraw love tokens from contract
- [x] User should be able to delegate voting rights to its delegatee (plain or with signature)

MasterChef

- [x] Should be able to enter/leave staking

RevenuePool

- [x] should not be able to deposit/withdraw prior to initialization
- [x] should initialize
- [x] should be able to deposit in the staking period
- [x] should not deposit in deposit delay
- [x] staking period should change after period end time
- [x] owner should be able to stop rewards any time
- [x] pending rewards should be accurate
- [x] Users should be able to withdraw
- [x] user can emergency withdraw to withdraw all staked tokens losing pending rewards
- [x] owner should be able to recover tokens other than stakedToken
- [x] owner should be able to emergency withdraw all the rewards

Timelock

- [x] Admin should be able to set pending admin only once
- [x] Admin should be able to queue/cancel/execute transactions

TokenTimelock

- ☑ release Time should be 3 years ahead of start time
- ☑ should not release any tokens before first year
- ☑ should release only 10% of token balance after first year
- ☑ should not release any tokens before second year (if first year tokens are already released)
- ☑ should release only 20% of token balance after second year(prior to release time)
- ☑ should not release any tokens before release time(if first and second year tokens are released)
- ☑ should release remaining token balance after release time

Closing Summary

Several issues of High, Medium and Low severity were found. All of the high and medium severity issues have been fixed. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Lovechain contract. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Lovechain team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

TheLoveChain

is forever

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com