Audit Whale

Smart contract code review and security analysis report

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Ellipsis Finance.
Approved by	Jameson| COO Audit Whale
Type	Multiple purposes contracts
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/ellipsis-finance/ellipsis/
Commit	
Deployed contract	
Timeline	30 MAR 2021 – 1 APR 2021
Changelog	1 APR 2021 – INITIAL AUDIT

Table of contents

Introduction

Audit Whale (Consultant) was contracted by Ellipsis Finance (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between March 30th, 2021 – April 1st, 2021.

Scope

The scope of the project is smart contracts in the repository:

Repository: https://github.com/ellipsis-finance/ellipsis/
Files:
EpsStaker.sol
FeeConverter.sol
LpTokenStaker.sol
MerkleDistributor.sol

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

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts have some issues that should be fixed.

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **0** critical, **2** high, **3** medium, **1** low, and **2** informational issues during the audit.

All issues have been addressed/accepted by the Customer.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

EpsStaker.sol

Description

EpsStaker is a contract which allows users to stake an ERC20 Token BaseToken, and as a reward for staking their token, they are offered users multiple different tokens

Imports

EpsStaker contract has the following imports:

1. openzeppelin/contracts/token/ERC20/IERC20.sol
2. openzeppelin/contracts/token/ERC20/SafeERC20.sol
3. openzeppelin/contracts/math/Math.sol
4. openzeppelin/contracts/math/SafeMath.sol
5. openzeppelin/contracts/access/Ownable.sol
6. openzeppelin/contracts/utils/ReentrancyGuard.sol

Inheritance

EpsStaker contract inherits from ReentrancyGuard and Ownable.

Usages

EpsStaker contract has the following usages:

1. SafeMath for uint256
2. SafeERC20 for IERC20
3. SafeERC20 for IMintableToken

Structs

EpsStaker contract has the following structs:

- Reward - Which contains information on various rewards per token stored
- Balances - Which contains information about locked rewards and earnings
- LockedBalance - Which contains information about unlocking time and amounts
- RewardData - Which contains token and amount data

Enums

EpsStaker contract has no custom enums.

Events

EpsStaker contract has following events:

- Staked - When an individual has staked a token
- Withdrawn - When an individual has withdrawn a staked token
- RewardPaid - When a reward has been paid based upon the staked token
- RewardsDurationUpdated - When the duration of awards distribution has been changed
- Recovered - When funds from LP rewards from other systems are distributed to holders

Modifiers

EpsStaker has one modifier, updateReward.

Fields

EpsStaker contract has the following fields:

- IMintableToken public stakingToken - Interface for the staking token
- address[] public rewardTokens - Array of reward tokens
- mapping(address => Reward) public rewardData - Mapping of addresses to rewards
- uint256 public constant rewardsDuration - Duration that rewards are streamed over
- uint256 public constant lockDuration - Duration of lock/earned penalty period
- mapping(address => bool) public minters - Addresses approved to call mint
- mapping(address=> mapping(address => bool)) public rewardDistributors
- // user -> reward token -> amount
- mapping(address => mapping(address => uint256)) public userRewardPerTokenPaid - Mapping of user rewards per token
- mapping(address => mapping(address => uint256)) public rewards - Mapping of user rewards

- uint256 public totalSupply - Total supply
- uint256 public lockedSupply - Presently locked supply
- mapping(address => Balances) balances - Private mapping for balance data
- mapping(address => LockedBalance[]) userLocks - Private mapping for balance data
- mapping(address => LockedBalance[]) userEarnings - Private mapping for balance data

Functions

EpsStaker has the following public functions:

- ***addReward***
  Description
  Adds a new reward token to be distributed to stakers
  Visibility
  public
  Input parameters
  - address _rewardsToken
  - address _distributor

  Constraints
  - rewardData[_rewardsToken].lastUpdateTime must not be 0

  Events emit
  No events are emit
  Output
  None
- ***stake***
  Description
  Stake tokens to receive rewards. Locked tokens cannot be withdrawn for lockDuration and are eligible to receive stakingReward rewards
  Visibility
  external

Input parameters

- uint256 amount
- bool lock

Constraints

- Amount must be greater than 0

Events emit

The staked event is emit

Output

None

- ***mint***

Description

Minted tokens receive rewards normally but incur a 50% penalty when withdrawn before lockDuration has passed.

Visibility

external

Input parameters

- address user
- uint256 amount

Constraints

- Amount must be greater than 0

Events emit

The staked event is presently emit. This should be corrected.

Output

None

- ***withdraw***

Description

The function first withdraws unlocked tokens, then earned tokens. Withdrawing earned tokens incurs a 50% penalty which is distributed based on locked balances.

Visibility

public

Input parameters

- uint256 amount

Constraints

- Amount must be greater than 0

Events emit

The withdrawn event is emit

Output

None

- ***getReward***

Description

The function first withdraws unlocked tokens, then earned tokens. Withdrawing earned tokens incurs a 50% penalty which is distributed based on locked balances.

Visibility

public

Input parameters

- uint256 amount

Constraints

- Amount must be greater than 0

Events emit

The withdrawn event is emit

Output

None

- ***getReward***

Description

The function first withdraws unlocked tokens, then earned tokens. Withdrawing earned tokens incurs a 50% penalty which is distributed based on locked balances.

Visibility

public

Input parameters

- uint256 amount

Constraints

- Amount must be greater than 0

Events emit

The withdrawn event is emit

Output

None

- ***exit***

  Description

  Withdraw full unlocked balance and claim pending rewards

  Visibility

  external

  Input parameters

  - msg.sender

  Constraints

  - Amount must be greater than 0

  Events emit

  The withdrawn event is emit

  Output

  None

- ***withdrawExpiredLocks***

  Description

  Withdraw all currently locked tokens where the unlock time has passed

  Visibility

  external

  Input parameters

  There are no input parameters

  Constraints

  There are no constraints

  Events emit

  The withdrawn event is emit

  Output

  None

_rewardPerToken, _earned, lastTimeRewardApplicable, rewardPerToken, getRewardForDuration, claimableRewards, totalBalance, unlockedBalance, earnedBalances, lockedBalances, withdrawableBalance

Description
Simple view functions.

FeeConverter.sol

Description

FeeConverter allows the conversion between two assets using the StableSwap as defined within StableSwap.vy (out of scope for this audit)

Imports

FeeConverter has no imports.

Inheritance

FeeConverter has no inheritance.

Usages

FeeConverter contract has the following usages:

1. SafeERC20 for IERC20

Structs

FeeConverter contract has no structs.

Enums

FeeConverter contract has no custom enums.

Events

FeeConverter contract emits no events.

Modifiers

FeeConverter has no modifiers.

Fields

FeeConverter has one field, address public feeDistributor.

Functions

FeeConverter has the following public functions:

- ***setFeeDistributor***
  Description
  Sets the new fee distributor
  Visibility
  external
  Input parameters
  - address distributor

  Constraints
  - Distributor address must not be 0

  Events emit
  No events are emit
  Output
  None

- ***convertFees***
  Description
  Execute a StableSwap between input and output coins
  Visibility
  external
  Input parameters
  - uint i - Amount of input coin
  - uint j - Amount of output coin

  Constraints
  No constraints.
  Events emit
  No events are emitted.
  Output
  None

- ***notify***
  Description

  Begins fee distribution on a given ERC20

  Visibility

  external

  Input parameters

  - IERC20 coin - The token upon which to begin the fee distribution

  Constraints

  No constraints.

  Events emit

  No events are emitted.

  Output

  None

LpTokenStaker.sol

Description

LpTokenStaker contract allows for LP tokens to be staked in order to generate EPS, Ellipsis' value-capture token

Imports

LpTokenStaker has the following imports:

1. contracts/token/ERC20/IERC20.sol
2. contracts/token/ERC20/SafeERC20.sol
3. contracts/math/SafeMath.sol
4. contracts/access/Ownable.sol

Inheritance

LpTokenStaker contract inherits Ownable.

Usages

LpTokenStaker contract has the following usages:

1. SafeMath for uint256
2. SafeERC20 for IERC20

Structs

LpTokenStaker contract has the following structs:

1. UserInfo - Info of each user.
2. PoolInfo - Info of each pool
3. EmissionPoint - Info about token emissions for a given time period

Enums

LpTokenStaker contract has no custom enums.

Events

LpTokenStaker contract has the following events:

1. Deposit - When a user deposits
2. Withdraw - When a user withdraws
3. EmergencyWithdraw - When a user withdraws without any rewards

Modifiers

LpTokenStaker has no modifiers.

Fields

LpTokenStaker has multiple fields:

- Minter public rewardMinter - Holds the minter identity
- uint256 public rewardsPerSecond - The number of rewards provisioned per second
- PoolInfo[] public poolInfo - Info of each pool.

- EmissionPoint[] public emissionSchedule - Data about the future reward rates. emissionSchedule stored in reverse chronological order, whenever the number of blocks since the start block exceeds the next block offset a new reward rate is applied.
- mapping(uint256 => mapping(address => UserInfo)) public userInfo - Info of each user that stakes LP tokens.
- uint256 public totalAllocPoint - Total allocation points. Must be the sum of all allocation points in all pools
- uint256 public startTime - The block number when reward mining starts
- Oracle[] public oracles - List of Chainlink oracle addresses.

Functions

LpTokenStaker has the following public functions:

- ***constructor***
  Description

  Initializes the function

  Visibility

  public

  Input parameters

  - uint128[] memory _startTimeOffset - Offset from the start time for reward distribution
  - uint128[] memory _rewardsPerSecond - Rewards that are distributed per second
  - IERC20 _fixedRewardToken - References the token itself being distributed

  Constraints

  No constraints exist.

  Events emit

  No events are emit

  Output

None

- ***deposit***
  Description

  Deposit LP tokens into the contract. Also triggers a claim.

  Visibility

  public

  Input parameters

  - uint256 _pid - Pool ID
  - uint256 _amount - Amount being deposited

  Constraints

  Amount must be greater than zero.

  Events emit

  A deposit event is emit upon successful deposit.

  Output

  None

- ***withdraw***
  Description

  withdraw LP tokens from the contract. Also triggers a claim.

  Visibility

  public

  Input parameters

  - uint256 _pid - Pool ID
  - uint256 _amount - Amount being deposited

  Constraints

  Amount must be greater than zero. Pool ID must be greater than zero.

Events emit

A withdraw event is emitted upon successful withdrawal.

Output

None

None

- ***emergencyWithdraw***
  Description

  withdraw LP tokens from the contract. Also triggers a claim.

  Visibility

  public

  Input parameters

  - uint256 _pid - Pool ID

  Constraints

  Pool ID must be greater than zero.

  Events emit

  An emergency withdraw event is emitted upon successful withdrawal.

  Output

  None

- ***claim***
  Description

  Claim pending rewards for one or more pools. Rewards are not received directly, they are minted by the rewardMinter.

Visibility

public

Input parameters

- uint256[] calldata _pids - Pool Ids from where to claim rewards

Constraints

List of pool IDs must be larger than zero.

Events emit

No event is emit.

Output

None

claimableReward, poolLength

Description
Simple view functions.

MerkleDistributor.sol

Description

MerkleDistributor allows for ongoing EPS airdrop to veCRV holders

Imports

MerkleDistributor contract has no imports

Inheritance

MerkleDistributor contract does not inherit from other contracts.

Usages

MerkleDistributor contract has no usages.

Structs

MerkleDistributor contract has following data structures:

- Claimed - Holds the account address information within the merkle tree

Enums

MerkleDistributor contract has no enums

Events

MerkleDistributor emits no events.

Modifiers

MerkleDistributor has no custom modifiers.

Fields

MerkleDistributor contract has following constants and fields:

- bytes32[] public merkleRoots - Byte array of presently existing merkleRoots
- bytes32 public pendingMerkleRoot - Merkle root that is waiting to be added
- uint256 public lastRoot - The most recently added root
- address public proposalAuthority - admin address which can propose adding a new merkle root
- address public reviewAuthority - admin address which approves or rejects a proposed merkle root
- mapping(uint256 => mapping(uint256 => uint256)) private claimedBitMap - Packed array of booleans of which can claim
- Minter public rewardMinter - References the instance of the minter which mints the rewards

Functions

MerkleDistributor has following public functions:

- ***constructor***
  Description

Initializes the function

Visibility

public

Input parameters

- address _proposalAuthority - admin address which can propose adding a new merkle root
- address _reviewAuthority - admin address which approves or rejects a proposed merkle root

Constraints

No constraints exist.

Events emit

No events are emit

Output

None

- ***setMinter***

  Description

  Sets which entity performs the minting

  Visibility

  public

  Input parameters

  - Minter _rewardMinter - The minter which performs the minting during value exchange

  Constraints

  - Minter must not be the 0 address

  Events emit

  No events are emitted.

  Output

  None

- ***setProposalAuthority***

  Description

  Sets the proposal authority address

  Visibility

  public

Input parameters

- address _account - The account to be assigned authority

Constraints

- msg.sender must presently be the proposal authority

Events emit

No event is emitted.

Output

None

- ***setReviewAuthority***

Description

Sets the review authority address

Visibility

public

Input parameters

- address _account - The account to be assigned authority

Constraints

- msg.sender must presently be the review authority

Events emit

No event is emitted.

Output

None

- ***proposewMerkleRoot***

Description

Each week, the proposal authority calls to submit the merkle root for a new airdrop.

Visibility

public

Input parameters

- bytes32 _merkleRoot - The root being proposed

Constraints

- msg.sender must be a present proposal authority
- Pending merkle root must be 0x00
- Total amount of merkle roots must be less than 52
- A week must have passed since the last addition.

Events emit
No events are emitted.
Output
None

- ***reviewPendingMerkleRoot***
  Description
  After validating the correctness of the pending merkle root, the reviewing authority calls to confirm it and the distribution may begin.
  Visibility
  public
  Input parameters
    - bool _approved - The present approval state of the root

  Constraints
    - msg.sender must be a review authority
    - Pending merkle root must not be the 0x00 address

  Events emit
  No events are emitted.
  Output
  None
- ***isClaimed***
  Description
  Assesses whether or not a given merkle index is claimed
  Visibility
  public
  Input parameters
    - uint256 merkleIndex - The present index on the merkle tree
    - uint256 index - The merkleDistributor index to be claimed

  Constraints
  There are no constraints present.
  Events emit
  There are no events emitted.
  Output
  A boolean value is returned as to whether or not the given index has been claimed yet per the claimed bitmap.
- ***claim***
  Description

Allows rewards to be claimed from a given index

Visibility

public

Input parameters

- uint256 merkleIndex - The index on the merkle tree
- uint256 index - The merkleDistributor index to be claimed
- uint256 amount - The amount to be claimed
- bytes32[] calldata merkleProof - The Merkle proof to be assessed before claiming

Constraints

- The merkle index must not exceed the length of the merkle tree
- The merkle index must not be already claimed.

Events emit

A claim event is emitted on successful claim and mint.

Output

None

Audit overview

Critical

1. No critical findings

High

1. LP token can be added more than once (no factor inhibits it from being added) which can result in rewards being misevaluated. It is suggested that a mapping of previously added LP tokens be used.

Customer Notice: We have mitigated this by placing the protocol behind a 3-of-5 multisig. The multisig means multiple people across multiple teams must sign off prior to the action, so that a duplicate token will not be added (either accidentally or maliciously).

2. The values for user.amount and user.rewardDebt are not set to zero until after the external call is made, which could lead to token theft if a malicious token is added.

Customer Notice: Adding a malicious token is mitigated by the use of a multisig requiring sign-off by multiple teams.

Medium

1. Compiler version used is too recent to be considered stable.

   We recommend updating to the latest stable one.

Customer Notice: We recognize and accept this risk.

2. If the first reward in rewardTokens is not the staking token, various contract functionality begins to decompose. As a result of such, constraints should be placed on the staking token (e.g. ensuring it is not the 0 address)

Customer Notice: We recognize and accept this risk.

3. Oracle validation within LpTokenStaker.sol involves calling the external contract (which may not be an Oracle), which could lead to abuse.

   Customer Notice: This is also mitigated by the use of the multisig.

- **Low**

1. Gas exhaustion may result in _massUpdatePools if too many pools have been added (partly depending upon present gas volumes)

   Customer Notice: We recognize and accept this risk.

- **Lowest / Code style / Best Practice**

1. Within MerkleDistribution.sol, *propose**w**MerkleRoot* presents a style-guide issue (w within the name denotes weekly, and therefore should be capitalized)
2. Magic numbers like 604800 should be declared constants

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in the As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **0** critical, **2** high, **3** medium, **1** low, and **2** informational issues during the audit.

All issues have been addressed/accepted by the Customer.

Notice: Multisig
https://bscscan.com/address/0x93E004080Fe6D967Eb01Bd294C8da12F970FeAb5

- Michael (Curve Finance Founder)
  0x425d16B0e08a28A3Ff9e4404AE99D78C0a076C5A
- Ben (Curve Finance CTO)
  0x7EeAC6CDdbd1D0B8aF061742D41877D7F707289a
- banteg (Yearn lead dev)
  0x7A1057E6e9093DA9C1D4C1D049609B6889fC4c67
- Alex (Ellipsis co-founder)
  0xbbcf44d219d57fd81771ec5053faea5fc8642193
- James (Ellipsis co-founder)
  0xabc00210a691ce0f3d7d0602d7d84aea4d91cdfd

Disclaimers

Audit Whale Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.