HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: GooseDeFi

Date: March 24th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 GooseDeFi.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, Governance, TimeLock, Defi
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/gooseadmin/goose-contracts-incubator
Commit	
Deployed contract	
Timeline	18 MAR 2021 – 24 MAR 2021
Changelog	24 MAR 2021 – INITIAL AUDIT

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by GooseDeFi (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 18th, 2021 – March 24th, 2021.

Scope

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

Contract deployment address:
Repository
File:

contracts/factories/FeeProcessorFactory.sol
contracts/factories/HouseFactory.sol
contracts/factories/IncubatorChefFactory.sol
contracts/factories/TokenFactory.sol
contracts/FeeProcessor.sol
contracts/HouseChef.sol
contracts/IncubatorChef.sol
contracts/LayerFactory.sol
contracts/GooseToken.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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 are medium secured.

Insecure	Poor secured	Secured	Well-secured

You are 

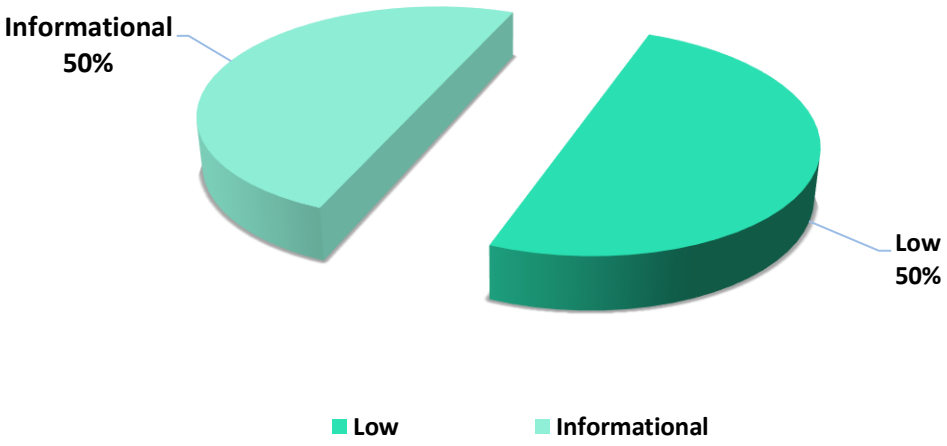
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section,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

Security engineers found **6** low, **6** informational issue during the audit.

Notice: the audit scope is limited and not include all files in the repository. Though, reviewed contracts are secure, we may not guarantee secureness of contracts that are not in the scope.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Graph 1. The distribution of vulnerabilities after the first review.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

AS-IS overview

FeeProcessor.sol

Description

FeeProcessor is a contract to store and work with fees.

Imports

FeeProcessor has following imports:

- import '@openzeppelin/contracts/access/Ownable.sol';
- import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
- import "@openzeppelin/contracts/math/SafeMath.sol";
- import "@openzeppelin/contracts/math/Math.sol";
- import "./interfaces/IPancakeRouter02.sol";
- import "./interfaces/IIncubatorChef.sol";
- import "./interfaces/IHouseChef.sol";
- import "./interfaces/IWETH.sol";
- import "./libs/PancakeLibrary.sol";
- import "./libs/IBEP20.sol";
- import "./libs/SafeBEP20.sol";
- import "./libs/BscConstants.sol";
- import "./interfaces/IFeeProcessor.sol";

Inheritance

FeeProcessor is Ownable, ReentrancyGuard, BscConstants, IFeeProcessor.

Usages

FeeProcessor contract has following usages:

- using SafeBEP20 for IBEP20;
- using SafeMath for uint256; **Structs**

Structs

FeeProcessor contract has no custom data structures.

Enums

FeeProcessor contract has no custom enums.

Events

FeeProcessor contract has folliwing custom events:

- event ProcessFees(address indexed user, address indexed token, uint256 amount);
- event ProcessSkipped(address indexed user, address indexed token, uint256 amount);
- event EmergencyWithdraw(address indexed user, address indexed token, uint256 amount);
- event SetFeeDevShare(address indexed user, uint16 feeDevShareBP);
- event SetSchedulerAddress(address indexed user, address newAddr);
- event ProcessorDeprecate(address indexed user, address newAddr);
- event SellTokens(address indexed user, address indexed token, uint256 amount);
- event BurnTokens(address indexed user, address indexed token, uint256 amount);
- event BuyGas(address indexed user, uint256 busdAmount, uint256 bnbAmount);
- event TaxGas(address indexed user, uint256 bnbAmount);

Modifiers

FeeProcessor has one custom modifiers:

- onlyAdmins.

Fields

FeeProcessor contract has following fields and constants:

- address public schedulerAddr;
- address public feeHolder;
- IBEP20 public gooseToken;
- IBEP20 public houseToken;
- IHouseChef public houseChef;
- IIncubatorChef public incubatorChef;

- uint16 public feeDevShareBP;
- uint16 public houseShareBP;
- uint16 public eggBuybackShareBP;
- //mapping(InputToken => mapping(OutputToken => path))
- mapping(address => mapping(address => address[])) paths;
- uint256 startTaxTimestamp = 0;
- uint256 taxedSinceStart = 0;
- uint256 constant maxGasTaxPerDay = 20 ether;

Functions

FeeProcessor has following public and external functions:

- ***constructor***
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
  - address _schedulerAddr,
  - address _gooseToken,
  - address _houseChef,
  - address _houseToken,
  - address _feeHolder,
  - uint16 _feeDevShareBP,
  - uint16 _houseShareBP,
  - uint16 _eggBuybackShareBP

  Constraints
  None
  Events emit
  None
  Output
  None
- ***receive***

Description
Empty overriding of receive function
Visibility
External payable
Input parameters
None
Constraints
None
Output
None

- ***setRouterPath, setIncubatorChef, sellTokens***
  Description
  Simple setters
  Visibility
  External
  Constraints
  OnlyAdmin
- ***processBusd***
  Description
  Function to process BUSD received as a fee.
  Visibility
  External
  Input parameters
  None
  Constraints
  onlyAdmins nonReentrant
  Output
  None
- ***taxGas***
  Description
  Tax some BNB for gas if Scheduler is running low
  Visibility
  External
  Input parameters
  None
  Constraints
  onlyAdmins nonReentrant
  Output
  None
- ***getBusdAmount***

Description
Function to return the amount of BUSD
Visibility
public view
Input parameters
- uint256 bnbAmount

Constraints
onlyAdmins nonReentrant
Output
- uint256

HouseChef.sol

Description

HouseChef – the modified version of PancakeSwap SmartChef. Reward tokens can be refilled by an internal call to refillRewards(). The pending rewards accumulation will pause if the reward balance dries out. This contract will be owned under Timelock.

Imports

HouseChef has following imports:

- import "@openzeppelin/contracts/math/SafeMath.sol";
- import "@openzeppelin/contracts/math/Math.sol";
- import "./libs/IBEP20.sol";
- import "./libs/SafeBEP20.sol";
- import "@openzeppelin/contracts/access/Ownable.sol";
- import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
- import "./interfaces/IWETH.sol";
- import "./interfaces/IHouseChef.sol";
- import "./libs/BscConstants.sol";

Inheritance

HouseChef is Ownable, ReentrancyGuard, IHouseChef, BscConstants.

Usages

HouseChef contract has following usages:

- SafeMath for uint256;
- SafeBEP20 for IBEP20;

Structs

HouseChef contract has following custom data structures:

- PoolInfo — contains data about lpPool
- UserInfo — contains data about user.

Enums

HouseChef contract has no custom enums.

Events

HouseChef contract has following custom events:

- event Harvest(address indexed user, uint256 amount);
- event Deposit(address indexed user, uint256 amount);
- event Withdraw(address indexed user, uint256 amount);
- event EmergencyWithdraw(address indexed user, uint256 amount);
- event UpdateEmissionRate(address indexed user, uint256 rewardsPerBlock);

Modifiers

HouseChef has no custom modifiers.

Fields

HouseChef contract has following fields and constants:

- IBEP20 public rewardToken;
- uint256 public rewardsPerBlock;
- PoolInfo[] public poolInfo;
- mapping(address => UserInfo) public userInfo;

- uint256 public totalAllocPoint = 0;
- uint256 public startBlock;

Functions

HouseChef has following public and external functions:

- ***constructor***
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
  - IBEP20 _stakeToken,
  - IBEP20 _rewardToken,
  - uint256 _rewardsPerBlock,
  - uint256 _startBlock

  Constraints
  None
  Events emit
  None
  Output
  None
- ***updateEmissionRate***
  Description
  Pancake has to add hidden dummy pools inorder to alter the emission, here we make it simple and transparent to all.
  Input parameters
  - *uint256 _rewardsPerBlock*

  Visibility
  public
  Constraints
  - *onlyOwner*

  Events emit
  None
  Output
  None
- ***harvestFor***
  Description

Function to trigger harvest for a specific user and pool. A specific user address is provided to facilitate aggregating harvests on multiple chefs.

Input parameters

- *address _user*

Visibility

public

Constraints

- *nonReentrant*

Events emit

None

Output

None

- ***pendingGoose***

Description

View function to see pending rewards on frontend.

Input parameters

- *address _user*

Visibility

external view

Constraints

None

Events emit

None

Output

uint256

- ***updatePool***

Description

Update reward variables of the given pool to be up-to-date.

Input parameters

None

Visibility

public

Constraints

None

Events emit

None

Output

None

- **refillRewards**

Description

Refill rewards into chef
Input parameters
uint256 _amount
Visibility
- external
- nonReentrant

Constraints
override
Events emit
None
Output
None

- **deposit**
  Description
  Refill rewards into chef
  Input parameters
  - uint256 _amount

  Visibility
  - public
  - nonReentrant

  Constraints
  - override

  Events emit
  None
  Output
  None
- **withdraw**
  Description
  Withdraw LP tokens from Chef.
  Input parameters
  - **uint256 _amount**

  Visibility
  - public

  Constraints
  - nonReentrant

  Events emit
  None
  Output
  None

- **emergencyWithdraw**
  Description
  Withdraw without caring about rewards.
  Input parameters
  None
  Visibility
  - public

  Constraints
  - nonReentrant

  Events emit
  None
  Output
  None

IncubatorChef.sol

Description

IncubatorChef – The modified version of GooseFinance MasterChef. On top of the deposit fee system that was introduced in Goose MasterChef, in this version, the staked amount is recorded into each PoolInfo to allow for adding multiple pools of the same stake token. A maximum deposit limit feature is also added to each pool. This contract will be owned under Timelock.

Imports

IncubatorChef has following imports:

- import "@openzeppelin/contracts/math/SafeMath.sol";
- import "@openzeppelin/contracts/math/Math.sol";
- import "./libs/IBEP20.sol";
- import "./libs/SafeBEP20.sol";
- import "@openzeppelin/contracts/access/Ownable.sol";
- import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
- import "./interfaces/IMintable.sol";
- import "./interfaces/IIncubatorChef.sol";

Inheritance

IncubatorChef is Ownable, ReentrancyGuard, IIncubatorChef.

Usages

IncubatorChef contract has following usages:

- SafeMath for uint256;
- SafeBEP20 for IBEP20;

Structs

IncubatorChef contract has following custom data structures:

- PoolInfo — contains data about lpPool
- UserInfo — contains data about user.

Enums

IncubatorChef contract has no custom enums.

Events

IncubatorChef contract has following custom events:

- event Harvest(address indexed user, uint256 indexed pid, uint256 amount);
- event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
- event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
- event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);
- event SetFeeAddress(address indexed user, address indexed newAddress);
- event SetDevAddress(address indexed user, address indexed newAddress);
- event UpdateEmissionRate(address indexed user, uint256 goosePerBlock);

Modifiers

IncubatorChef has no custom modifiers.

Fields

IncubatorChef contract has following fields and constants:

- IMintable public goose;
- address public devAddress;
- uint256 public goosePerBlock;
- uint256 public constant BONUS_MULTIPLIER = 1;
- address public feeAddress;
- uint256 public maxGooseSupply;
- PoolInfo[] public poolInfo;
- mapping(uint256 => mapping(address => UserInfo)) public userInfo;
- uint256 public totalAllocPoint = 0;
- uint256 public startBlock;

Functions

IncubatorChef has following public and external functions:

- ***constructor***
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
  - IMintable _goose,
  - address _devAddress,
  - address _feeAddress,
  - uint256 _goosePerBlock,
  - uint256 _startBlock,
  - uint256 _maxGooseSupply IMintable _goose,
  - address _devAddress,
  - address _feeAddress,
  - uint256 _goosePerBlock,
  - uint256 _startBlock,

- uint256 _maxGooseSupply

Constraints
None
Events emit
None
Output
None

- ***poolLength***
Description
Add a proposal ID to a current epoch.
Visibility
external view
Input parameters
None
Constraints
None
Events emit
None
Output
uint256
- ***add***
Description
Add a new lp to the pool. Can only be called by the owner.
Visibility
override external
Input parameters
  - uint256 _allocPoint,
  - IBEP20 _lpToken,
  - uint16 _depositFeeBP,
  - uint256 _maxDepositAmount,
  - bool _withUpdate

Constraints
onlyOwner
Events emit
None
Output
None

- ***set***
  Description
  Update the given pool's GOOSE allocation point and deposit fee. Can only be called by the owner.
  Visibility
  override external
  Input parameters
  - uint256 _pid,
  - uint256 _allocPoint,
  - uint16 _depositFeeBP,
  - uint256 _maxDepositAmount,
  - bool _withUpdate

  Constraints
  onlyOwner
  Events emit
  None
  Output
  None
- ***emergencyWithdraw***
  Description
  Withdraw without caring about rewards.
  Visibility
  external
  Input parameters
  - uint256 _pid,

  Constraints
  nonReentrant
  Events emit
  None
  Output
  None
- ***withdraw***
  Description
  Withdraw LP tokens from MasterChef.
  Visibility
  external
  Input parameters

- uint256 _pid,
- uint256 _amount

Constraints
nonReentrant
Events emit
None
Output
None

- ***harvestFor***
  Description
  New function to trigger harvest for a specific user and pool. A specific user address is provided to facilitate aggregating harvests on multiple chefs
  Visibility
  public
  Input parameters
  - uint256 _pid,
  - address _user

  Constraints
  nonReentrant
  Events emit
  None
  Output
  None
- ***bulkHarvestFor***
  Description
  New function to trigger harvest for a specific user and pool. A specific user address is provided to facilitate aggregating harvests on multiple chefs
  Visibility
  external
  Input parameters
  - uint256[] calldata pidArray,
  - address _user

  Constraints
  nonReentrant
  Events emit
  None
  Output
  None

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

LayerFactory.sol

Description

LayerFactory – This contract contains the logic to create a new layer, including deploying new contracts. An external scheduler will be calling the createNewLayer() function every X days. The scheduler will also be adding the farms by calling addPool(), and finally passing ownership to timelock afterwards by calling startTimelock().

Imports

LayerFactory has following imports:

- import "./libs/IBEP20.sol";
- import "./IncubatorChef.sol";
- import "@openzeppelin/contracts/access/Ownable.sol";
- import "@openzeppelin/contracts/math/SafeMath.sol";
- import "@openzeppelin/contracts/math/Math.sol";
- import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
- import "./interfaces/IPancakeRouter02.sol";
- import "./libs/PancakeLibrary.sol";
- import "./interfaces/IMintable.sol";
- import "./interfaces/ITokenFactory.sol";
- import "./interfaces/IHouseFactory.sol";
- import "./interfaces/IFeeProcessorFactory.sol";
- import "./interfaces/IIncubatorChefFactory.sol";
- import "./interfaces/IIncubatorChef.sol";
- import "./interfaces/IFeeProcessor.sol";
- import "./libs/BscConstants.sol";
- import "./FeeProcessor.sol";

Inheritance

LayerFactory is Ownable, BscConstants.

Usages

LayerFactory contract has following usages:

- SafeMath for uint256;
- SafeBEP20 for IBEP20;

Structs

LayerFactory contract has following custom data structures:

- LayerInfo — contains data about each layer

Enums

LayerFactory contract has no custom enums.

Events

LayerFactory contract has following custom events:

- event CreateNewLayer(address indexed user, uint256 indexed layerId);
- event AddLiquidity(uint256 indexed layerId, address indexed gooseToken, address indexed priceToken, uint256 gooseAmount, uint256 priceAmount);
- event RemoveLiquidity(uint256 indexed layerId, address indexed gooseToken, address indexed priceToken, uint256 lpAmount);
- event BurnToken(uint256 indexed layerId, address indexed gooseToken, uint256 gooseAmount);
- event SetFeeHolder(address indexed user, address feeHolder);
- event SetGooseHolder(address indexed user, address gooseHolder);
- event SetScheduler(address indexed user, address scheduler);
- event StartTimelock(address indexed user, uint256 indexed layerId);
- event UpdateFactoryAddresses(address indexed user);
- event UpdateNewLayerSettings(address indexed user);
- event FundsWithdraw(address indexed user, address indexed token, uint256 amount);

Modifiers
LayerFactory has no custom modifiers.

Fields

LayerFactory contract has following fields and constants:

- LayerInfo[] public layers;
- address public timelock;
- address public schedulerAddr;
- address public gooseHolder;
- address public feeHolder;
- ITokenFactory public tokenFactory;
- IHouseFactory public houseFactory;
- IFeeProcessorFactory public feeProcessorFactory;
- IIncubatorChefFactory public incubatorChefFactory;
- uint256 public tokenPerBlock = 1 ether;
- uint256 public maxTokenSupply = 1000000 ether;
- IBEP20 public houseToken = IBEP20(busdAddr);
- uint256 public houseEmitRate = 1.5 ether;
- uint256 public totalMint = 20000 ether;
- uint256 public pricePerGoose = 10 ether;
- uint16 public feeDevShareBP = 1000;
- uint16 public houseShareBP = 3000;
- uint16 public eggBuybackShareBP = 3000;
- address[] busdToBnbPath = [busdAddr, wbnbAddr];

Functions

LayerFactory has following public and external functions:

- ***constructor***
  Description
  Initializes the contract.
  Visibility
  public

Input parameters

- address _schedulerAddr,
- address _feeHolder,
- address _gooseHolder,
- address _timelock,
- address _tokenFactory,
- address _houseFactory,
- address _feeProcessorFactory,
- address _incubatorChefFactory

Constraints
None
Events emit
None
Output
None

- ***createNewLayer***

Input parameters
None
Visibility
external
Constraints

- onlyAdmins

Events emit
None
Output
None

- ***createNewToken***

Input parameters

- uint256 layerId,
- string calldata tokenName,
- string calldata tokenSymbol

Visibility
external
Constraints

- onlyAdmins

Events emit

None
Output
None

- ***createNewToken***
  Input parameters
  - uint256 layerId,
  - string calldata tokenName,
  - string calldata tokenSymbol

  Visibility
  external
  Constraints
  - onlyAdmins

  Events emit
  None
  Output
  None
- ***createNewFeeProcessor, createNewHouse, createNewIncubatorChef***
  Description
  Functions to add some configuration factory parameters
  Visibility
  external
  Constraints
  - onlyAdmins

  Events emit
  None
  Output
  None
- ***removeWBNBLiquidity, removeBUSDLiquidity***
  Description
  Functions to manage tokens
  Visibility
  external
  Constraints
  - onlyAdmins

  Events emit

None
Output
None

- ***setPool, massUpdatePools, addPool***
  Description
  Functions to manage lpPools in chef contract
  Visibility
  external
  Constraints
  - onlyAdmins

  Events emit
  None
  Output
  None
- ***startTimelock***
  Description
  Transfer ownership of IncubatorChef and HouseChef to timelock
  Visibility
  external
  Constraints
  - onlyAdmins

  Events emit
  None
  Output
  None
- ***setFeeHolder,***
  setSchedulerAddr,
  setGooseHolder,
  updateFeeProcessorAddress,
  updateFactoryAddresses,
  updateNewLayerSettings
  Description
  Functions to manage some factory configurations
  Visibility
  external
  Constraints
  - onlyAdmins

  Events emit
  None
  Output

None

- ***fundsWithdraw***
  Description
  Withdraw any excess funds used for providing initial LP
  Visibility
  external
  Constraints
  - onlyAdmins

  Events emit
  None
  Output
  None

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

No critical issues were found.

High

No critical issues were found.

Medium

No Medium issues were found.

Low

1. Despite the contract works with Bep20, which can't make recurrent function calling in the scope of token transfer operations, it's much better to avoid situations, where only protocol inability to make actions protects you. Otherwise, the user can call the deposit function from an address with an overridden fallback function. So that, user can call the deposit function and get rewards each time he wants.
   Contract: IncubatorChef.sol

   Recommendation: Move the calculation of the user.rewardDebt new value to the start of the pending reward transferring.

2. Despite the contract works with Bep20, which can't make recurrent function calling in the scope of token transfer operations, it's much better to avoid situations, where only protocol inability to make actions protects you. Otherwise, the user can call the withdraw function from an address with an overridden fallback function. So that, user can call the deposit function and get rewards each time he wants.
   Contract: IncubatorChef.sol

   Recommendation: Move the calculation of the user.rewardDebt new value to the start of pending the reward transferring.

3. Despite the contract works with Bep20, which can't make recurrent function calling in the scope of token transfer operations, it's much better to avoid situations, where only protocol inability to make actions protects you. Otherwise, the can call the withdraw function from an address with

an overridden fallback function. So that, user can call the deposit function and get rewards each time he wants.
Contract: HouseChef.sol
Recommendation: Move the calculation of the user.rewardDebt new value to the start of the pending reward transferring.

4. Despite the contract works with Bep20, which can't make recurrent function calling in the scope of token transfer operations, it's much better to avoid situations, where only protocol inability to make actions protects you. Otherwise, the user can call the deposit function from an address with an overridden fallback function. So that, user can call the deposit function and get rewards each time he wants.
   Contract: HouseChef.sol

   Recommendation: Move the calculation of the user.rewardDebt new value to the start of pending reward transferring.

5. There is a set of cases when the deposit fee will be equal to zero in IncubatorChef.deposit() and HouseChef.deposit() functions. Also, it is redundant to execute a function, when the amount is equal to zero.

   Contract: HouseChef.sol, IncubatorChef.sol

   Recommendation: Set the minimum deposit amount.

6. There is a probability to except calling some of the LayerFactory functions and provoke invalid logic execution.

   Contract: LayerFactory.sol

   Recommendation: Use the "Factory Method" design pattern or implement the "Factory" pattern correctly.

Lowest / Code style / Best Practice

1. Some code style issues were found by the static code analyzers.

2. It is a good practice not to hardcode values in code. Extract hardcodes to the constant fields in FeeProcessor contract.

   Contract: FeeProcessor.sol

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

3. It is redundant to recalculate user.rewardDebt if deposit amount is equal to 0 in IncubatorChef.deposit, HouseChef.deposit, IncubatorChef.withdraw and HouseChef.withdraw functions. Bad code style.

   Contract: IncubatorChef.sol, HouseChef.sol

   Recommendations: create a new function updatePool(uint256 _pid) with functionality to update pool data and user reward debt.

4. It is redundant to recalculate the user.rewardDebt if the withdrawal amount is equal to 0 in IncubatorChef.withdraw and HouseChef.withdraw functions. Bad code style.

   Contract: IncubatorChef.sol, HouseChef.sol

   Recommendations: create a new function updatePool(uint256 _pid) with functionality to update pool data and user reward debt.

5. There is a validation that the msg.sender or msg.origin is equal to harvest target. Create IncubatorChef.harvest(uint256 _pid) and HouseChef.harvest(uint256 _pid) and replace calling harvestFor() function with them, where the second parameter will be equal msg.sender. These functions will be much easy to use. Moreover, there is a validation that msg.sender or tx.origin is equal to harvest target. So only the function caller can be targeted by harvest.

   Contract: IncubatorChef.sol, HouseChef.sol

   Recommendations: create a new function updatePool(uint256 _pid) with functionality to update pool data and user reward debt.

6. It will be much better to extract duplicated code to separate function in IncubatorChef, HouseChef due to code style.

   uint256 pending = user.amount.mul(pool.accGoosePerShare) .div(1e12).sub(user.rewardDebt);

   if (pending > 0) {

   safeGooseTransfer(msg.sender, pending);

   }

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 As-Is overview section of the report.

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

Security engineers found **6** low, **6** informational issue during the audit.

Notice: the audit scope is limited and not include all files in the project. Though, reviewed contracts are secure, we may not guarantee secureness of codebase that are not in the scope.

Violations in the following categories were found and addressed to Customer:

Category	Check Item
Code review	▪ Code style
	▪ Lack of validation

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken 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.