CERTIK

Security Assessment

ButterSwap II

Jul 13th, 2021

Table of Contents

Appendix

Disclaimer

About

Summary

This report has been prepared for ButterSwap II to discover issues and vulnerabilities in the source code of the ButterSwap II project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Additionally, this audit is based on a premise that all external contracts were implemented safely.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	ButterSwap II
Platform	Heco
Language	Solidity
Codebase	https://github.com/butter-swap/butter-swap-farm/tree/master/contracts
Commit	8c16ba093a2eb401e0955674a7eaed05ad4b6b90

Audit Summary

Delivery Date	Jul 13, 2021
Audit Methodology	Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	5	0	0	0	5	0
Informational	21	0	1	7	2	11
Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
BCB	contracts/BoardChef.sol	c3695c0ec5b59fc601d3c9f607d9a8988ced7c3c86274911c4e04e63091482db
BTB	contracts/BoardToken.sol	3145122df2b7e4233db3ba07b9c85afade5c1ee5fd02fb362a03171e2d83b66b
BDB	contracts/ButterDao.sol	39ee6fa3c5af743554c7528fb4a609f6bb64cbcb8bfaaaf7ae61b61633aa5e4f
BVB	contracts/ButterVault.sol	392d5637c3449a2039a266e28d30a8ec4a1a364613357aad828538dcb1444ab3
DAO	contracts/DAOToken.sol	d38c8e6a01a5f167e393054ce3e31f2acb691ce4eb013624e96fd62b93848cf5
IBD	contracts/IButterDao.sol	765747b818a615d42722fd2919bf44afd5b6ac1971aee367a0ef8bfee82804dd
ILL	contracts/ILuckyLucky.sol	556968c6bd5eaf8c89b4d62b7daa71924cdedc406b7ea6ffef177c2b89f869cb
IMC	contracts/IMasterChef.sol	ba461eb9f575f72e5bda17ecaf9a09fb2243c2a7dfdd00aead45927a0c533004
IRN	contracts/IRandomNumberGenerator.sol	bcc2ea4ccc78794fdd1f8b849c5a271d2e00da2a3db510d124dda0fa61cba79e
LLC	contracts/LuckyLuckyChef.sol	a928fd801fbb2dc489c14b80a25a3d507775c83db942d431650cb0c84abe3fae
PBS	contracts/Pausable.sol	ea6e62a6711763fe9afbfec47c7f9d21f447e31967212bc92caf272810d5e756
RNG	contracts/RandomNumberGenerator.sol	660ca0f92d6721dff2e072c3cff71803c964cec8d42dc6ea4caa76d497dcec70

Understandings

Overview

The `BoardToken` is a standard HRC20 contract, the owner of contract can `mint` tokens to any account and `burn` tokens from any account.

The `BoardChef` is a mining contract, users can stake `boardToken` to obtain reward token.

The `DAOToken` is a standard HRC20 contract, the owner of contract can `mint` tokens to any account and `burn` tokens from any account. When users get `daoToken`, they will get the same amount of delegated voting which allows them to participate in community governance. When they transfer `daoToken` to other accounts, they will lose the same amount of delegated voting.

In the `ButterDao` contract, users can stake `creamToken`, the staked amount must be larger than 0.1% of butter total supply at the first time. After users have staked, they can obtain the same amount of `daoToken` and `boardToken` and become the member of `daoMembers`. Similarly, users can use the same amount of `daoToken` and `boardToken` to exchange `creamToken` that they staked. If users transfer some `daoToken` or `boardToken` to others, the same amount of `creamToken` cannot be withdrawn. Users can only call `emergencyWithdraw` function to withdraw the `creamToken` with the same amount of `daoToken` and `boardToken` they had.

In the `ButterVault` contract, users can deposit `butterToken` and obtain related shares. The contract will stake the tokens deposited by users to the `masterChef` contract. When users withdraw tokens, the contract will charge some fees. The fee ratio is different based on whether the user is a member of `daoMembers`. Members charge 0.1%, others charge 0.2%, 50% of the fees will be transferred to the dead address, and the rest will be transferred to the treasury.

Users can call the `harvest` function to extract the contract's mining revenue to `masterChef` contract, and the contract will transfers 5% of the mining revenue to users as the reward.

In the `LuckyLuckyChef` contract, the `admin` of the contract can start the lottery. During the activity, users can deposit `boardToken` to participate in the lottery. The contract will calculate `power` based on the amount deposited by users. `power = amount * (endBlock - startBlock)`. And the `admin` will randomly draw a lucky address based on users' `power` after the activity ends to win the reward. If users withdraw all the tokens, they will lose the chance of winning the lottery reward.

All the values mentioned above can be referenced only since they can be changed by the `owner` at any time.

Privileged Functions

The project contains the following privileged functions that are restricted by some modifiers. They are used to modify the contract configurations and address attributes. We grouped these functions below:

The `onlyOwner` modifier:

Contract `BoardChef`:

- function stopReward()
- function emergencyRewardWithdraw()

Contract `BoardToken`:

- function mint(address _to, uint256 _amount)
- function burn(address _from, uint256 _amount)

Contract `ButterDao`:

- function switchCondition(bool _turnOn)
- function changeThresholdDivider(uint256 _thresholdDivider)

Contract `DAOToken`:

- function mint(address _to, uint256 _amount)
- function burn(address _from, uint256 _amount)

Contract `ButterVault`:

- function setAdmin(address _admin)
- function setTreasury(address _treasury)
- function setBurnThreshold(uint256 _burnThreshold)

Contract `LuckyLuckyChef`:

- function setAdmin(address _admin)
- function updateRewardPerPeriod(uint256 _rewardPerPeriod)
- function withdrawRewardToken(uint256 _amount)

The `onlyAdmin` modifier:

Contract `ButterVault`:

- function setCallFee(uint256 _callFee)
- function setWithdrawFeePeriod(uint256 _withdrawFeePeriod)
- function setPerformanceFee(uint256 _performanceFee)
- function setWithdrawFee(uint256 _withdrawFee)
- function setWithdrawFeeBoard(uint256 _withdrawFeeBoard)
- function emergencyWithdraw()
- function inCaseTokensGetStuck(address _token)
- function pause()
- function unpause()

Contract `LuckyLuckyChef`:

- function startNewLucky(uint256 _endBlock)
- function finishLuckyInternal()
- function finishLucky(uint256 _seed)

The `whenNotPaused` modifier:

Contract `ButterVault`:

- function deposit(uint256 _amount)
- function harvest()
- function pause()

The `whenPaused` modifier:

Contract `ButterVault`:

- function unpause()

The `initializer` modifier:

Contract `LuckyLuckyChef`:

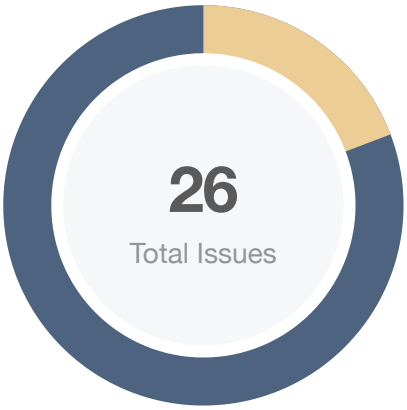
- function initialize(address _IRandomNumberGenerator)

The `onlyRandomGenerator` modifier:

Contract `LuckyLuckyChef`:

- function numbersDrawn(uint256 _totalPower, bytes32 _requestId, uint256 _randomNumber)

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Medium	0 (0.00%)
Minor	5 (19.23%)
Informational	21 (80.77%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
BCB-01	Lack Of Input Validation	Logical Issue	Informational	Declined
BCB-02	Meaningless Validation	Logical Issue	Informational	Declined
BCB-03	Privileged Ownership	**Centralization / Privilege**	**Minor**	**Acknowledged**
BCB-04	Missing Emit Events	Coding Style	Informational	Declined
BCB-05	Proper Usage of `public` And `external` Type	Gas Optimization	Informational	Resolved
BCB-06	Lack Of Judgment Conditions	Logical Issue	Informational	Resolved
BDB-01	Boolean Equality	Coding Style	Informational	Resolved
BDB-02	Code Reuse	Coding Style	Informational	Declined
BDB-03	Meaningless Calculation	Coding Style	Informational	Declined
BDB-04	Proper Usage of `public` And `external` Type	Gas Optimization	Informational	Partially Resolved
BDB-05	Missing Emit Events	Coding Style	Informational	Declined
BDB-06	Privileged Ownership	**Centralization / Privilege**	**Minor**	**Acknowledged**
BTB-01	Proper Usage of `public` And `external` Type	Gas Optimization	Informational	Declined

ID	Title	Category	Severity	Status
BTB-02	Privileged Ownership	**Centralization / Privilege**	**Minor**	**Acknowledged**
BVB-01	Missing Emit Events	Coding Style	Informational	Declined
BVB-02	Missing Zero Address Validation	Logical Issue	Informational	Resolved
BVB-03	Discussion For `withdraw` Function	Logical Issue	Informational	Acknowledged
DAO-01	Privileged Ownership	**Centralization / Privilege**	**Minor**	**Acknowledged**
LLC-01	Meaningless Validation	Logical Issue	Informational	Declined
LLC-02	Divide Before Multiply	Mathematical Operations	Informational	Resolved
LLC-03	Integer Overflow Risk	Mathematical Operations	Informational	Resolved
LLC-04	Missing Zero Address Validation	Logical Issue	Informational	Declined
LLC-05	Risk For Weak Randomness	Logical Issue	Informational	Acknowledged
LLC-06	Proper Usage of `public` And `external` Type	Gas Optimization	Informational	Resolved
LLC-07	Redundant Data	Logical Issue	Informational	Declined
LLC-08	Privileged Ownership	**Centralization / Privilege**	**Minor**	**Acknowledged**

BCB-01 | Lack Of Input Validation

Category	Severity	Location	Status
Logical Issue	Informational	contracts/BoardChef.sol: 59	Declined

Description

The given input `_boardToken`,`_rewardToken` is missing the sanity check for the non-zero address and `_startBlock`,`_bonusEndBlock` is missing the sanity check for the value size in the aforementioned line.

Recommendation

We recommend adding the check for the passed-in values to prevent unexpected error as below:
constructor():

```
require(_boardToken != address(0), "_boardToken address cannot be 0");
require(_rewardToken != address(0), "_rewardToken address cannot be 0");
require(_startBlock < _bonusEndBlock, "_startBlock must less than _bonusEndBlock");
```

Alleviation

No alleviation.

BCB-02 | Meaningless Validation

Category	Severity	Location	Status
Logical Issue	Informational	contracts/BoardChef.sol: 126, 150	Declined

Description

The uint256 is an unsigned integer, so the value of uint type is always greater than or equal to 0.

Recommendation

We recommend removing the validation.

Alleviation

No alleviation.

BCB-03 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	contracts/BoardChef.sol: 79, 181	**Acknowledged**

Description

The owner of contract `BoardChef` has the permission to:

1. stop mining immediately and no more rewards will be issued by `stopReward` function.
2. emergency withdrawal of rewards in the contract by `emergencyRewardWithdraw` function.

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

Customer team response:
DAO/governance/voting module will be introduced in the future.

BCB-04 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	contracts/BoardChef.sol: 79, 181	Declined

Description

Some functions should be able to emit event as notifications to customers because they change the status of sensitive variables.This suggestion applies to other similar codes.

Recommendation

Consider adding an emit after changing the status of variables.

Alleviation

No alleviation.

BCB-05 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/BoardChef.sol: 79, 125, 149, 171, 181	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

BCB-06 | Lack Of Judgment Conditions

Category	Severity	Location	Status
Logical Issue	Informational	contracts/BoardChef.sol: 84	Resolved

Description

Although the parameters passed in in the contract are correct, because the `getMultiplier` function can be called by external contracts, the parameters passed in from outside will result in incorrect results due to lack of judgment conditions.

Recommendation

We recommend modifying as below:

```
function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256)
{
        if (_to <= startBlock || _from >= bonusEndBlock) {
            return 0;
        } else if (_from <= startBlock && _to >=bonusEndBlock) {
            return bonusEndBlock.sub(startBlock);
        } else if (_from <= startBlock && _to > startBlock) {
            return _to.sub(startBlock);
        } else if (_from >= startBlock && _to <= bonusEndBlock) {
           return _to.sub(_from);
       } else {
            return bonusEndBlock.sub(_from);
       }
    }
```

or modify the function visibility from `public` to `internal`.

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

BDB-01 | Boolean Equality

Category	Severity	Location	Status
Coding Style	Informational	contracts/ButterDao.sol: 80, 121, 181	Resolved

Description

Boolean constants can be used directly and do not need to be compare to true or false.

Recommendation

We recommend removing the equality to the boolean constant.For example:
enterStake():

```
1      if (daoMembers[msg.sender]) {...}
```

leaveStake():

```
1      require(daoMembers[msg.sender],"leaveStake: you are not dao member");
```

leaveStakePrecheck():

```
1      if (!daoMembers[msg.sender]) {...}
```

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

BDB-02 | Code Reuse

Category	Severity	Location	Status
Coding Style	Informational	contracts/ButterDao.sol: 90, 163	Declined

Description

The code for calculating the `threshold` in the `enterStake` function is exactly the same as that in the `firstStakeThreshold` function. We recommend to reuse this part of the code to keep the code concise.

Recommendation

We recommend modifying as below:

enterStake():

```
    if (daoMembers[msg.sender]) {
        ...
    } else {
        uint256 threshold = firstStakeThreshold();
        ...
    }
```

firstStakeThreshold():

```
    function firstStakeThreshold() public view returns (uint256) {...}
```

Alleviation

No alleviation.

BDB-03 | Meaningless Calculation

Category	Severity	Location	Status
Coding Style	Informational	contracts/ButterDao.sol: 97, 172	Declined

Description

The decimals of `ButterToken` and `CreamToken` are both 18, so there is no need to calculate the accuracy range and it makes the calculation seems more redundant.

Recommendation

We recommend modifying as below:

```
    function firstStakeThreshold() external view returns (uint256) {
        ...
        uint256 threshold =validTotal.div(thresholdDivider);
        return threshold;
    }
```

Alleviation

No alleviation.

BDB-04 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/ButterDao.sol: 64, 69, 76, 119, 234	Partially Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

Alleviation

The team heeded some of our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

BDB-05 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	contracts/ButterDao.sol: 64, 68	Declined

Description

Some functions should be able to emit event as notifications to customers because they change the status of sensitive variables.This suggestion applies to other similar codes.

Recommendation

Consider adding an emit after changing the status of variables.

Alleviation

No alleviation.

BDB-06 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	contracts/ButterDao.sol: 119, 234	**Acknowledged**

Description

The owner of contract `ButterDao` has the permission to:

1. set whether to restrict users from withdrawing, if it is restricted, whether it is normal withdrawal or emergency withdrawal, users need to deposit for a period more than 7 days and can only withdraw on Sundays by `leaveStake` and `emergencyWithdraw` function.

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

Customer team response:
DAO/governance/voting module will be introduced in the future.

BTB-01 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/BoardToken.sol: 10, 14	Declined

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

Alleviation

No alleviation.

BTB-02 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	contracts/BoardToken.sol: 10, 14	**Acknowledged**

Description

The owner of contract `BoardToken` has the permission to:

1. mint token to account by `mint` function.
2. burn token from account by `burn` function.

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

Customer team response:
DAO/governance/voting module will be introduced in the future.

BVB-01 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	contracts/ButterVault.sol: 106, 115, 124, 133, 145, 154, 163, 172	Declined

Description

Some functions should be able to emit event as notifications to customers because they change the status of sensitive variables.This suggestion applies to other similar codes.

Recommendation

Consider adding an emit after changing the status of variables.

Alleviation

No alleviation.

BVB-02 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ButterVault.sol: 66	Resolved

Description

The given input is missing the sanity check for the non-zero address in the aforementioned line.

Recommendation

Consider adding a check like below:

constructor():

```
require(address(_token) != address(0), "_token address cannot be 0");
require(address(_receiptToken) != address(0), "_receiptToken address cannot be 0");
require(address(_masterchef) != address(0), "_masterchef address cannot be 0");
require(address(_butterDao) != address(0), "_butterDao address cannot be 0");
require(_admin != address(0), "_admin address cannot be 0");
require(_treasury != address(0), "_treasury address cannot be 0");
```

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

BVB-03 | Discussion For `withdraw` Function

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ButterVault.sol: 270	Acknowledged

Description

Under what circumstances will `diff < balWithdraw`? The `butterToken` deposited by the user will be staked by the contract to `masterChef` for mining to obtain rewards. The reward is also `butterToken`, and after the reward is withdrawn, it will be staked again to `masterChef`. The final balance in the contract should be greater than the amount deposited by the user.

Alleviation

Customer team response:
This is to prevent `MasterChef` contract upgrades in the future, `leaveStaking` function will charge fees, etc. This situation does not occur at present.

DAO-01 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	contracts/DAOToken.sol: 315, 319	**Acknowledged**

Description

The owner of contract `DAOToken` has the permission to:

1. mint token to account by `mint` function.
2. burn token from account by `burn` function.

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

Customer team response:
DAO/governance/voting module will be introduced in the future.

LLC-01 | Meaningless Validation

Category	Severity	Location	Status
Logical Issue	Informational	contracts/LuckyLuckyChef.sol: 351, 357, 366, 374, 378, 384	Declined

Description

The uint256 is an unsigned integer, so the value of uint type is always greater than or equal to 0. We recommend to modify the check to be `_amount>0` and remove the conditional judgment of `_amount>0` afterwards.

Recommendation

We recommend modifying the validation as below: deposit():

```
1      require (_amount > 0, 'amount cannot be 0');
```

withdraw():

```
1      require (_amount > 0, 'amount cannot be 0');
```

Remove the conditional judgment of `_amount>0`.

Alleviation

No alleviation.

LLC-02 | Divide Before Multiply

Category	Severity	Location	Status
Mathematical Operations	Informational	contracts/LuckyLuckyChef.sol: 381	Resolved

Description

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation

Consider ordering multiplication before division. For example:

```
1       user.power = user.power.mul(user.amount).div(formerAmount);
```

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

LLC-03 | Integer Overflow Risk

Category	Severity	Location	Status
Mathematical Operations	Informational	contracts/LuckyLuckyChef.sol: 359	Resolved

Description

Using `+` in the method directly to calculate the value of the variable may overflow. `SafeMath` provides a method to verify overflow, and it is safer to use the method provided.

Recommendation

Using the `add()` function in `SafeMath` library for mathematical operations. For example:

```
1      user.power = user.power.add(_amount.mul(endBlock.sub(block.number)));
```

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

LLC-04 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	contracts/LuckyLuckyChef.sol: 139, 166	Declined

Description

The given input is missing the sanity check for the non-zero address in the aforementioned line.

Recommendation

Consider adding a check like below:

constructor():

```
require(address(_board) != address(0), "_board address cannot be 0");
require(address(_rewardToken) != address(0), "_rewardToken address cannot be 0");
require(_admin != address(0), "_admin address cannot be 0");
```

setAdmin():

```
require(_admin != address(0), "_admin address cannot be 0");
```

Alleviation

No alleviation.

LLC-05 | Risk For Weak Randomness

Category	Severity	Location	Status
Logical Issue	Informational	contracts/LuckyLuckyChef.sol: 258, 284	Acknowledged

Description

The `sumLuckyPower` is obtained by encoding a random number with `block.timestamp` and `block.difficulty`, and then generating the remainder of `totalPower`. The values of `block.timestamp`,`block.difficulty` and `totalPower` can be queried, so we think the private variable `sumLuckyPower` based on inner operations can be predicted. If the parameter passed to `numbersDrawn` is not a random number, then the result is not a random number.

Recommendation

Consider obtained the `sumLuckyPower` based on a third-part random service such as chainlink(https://docs.chain.link/docs/get-a-random-number/).

Alleviation

Chainlink currently does not support the VRF function on the heco-chain, when it is supported, it will switch to the function of obtaining random numbers from the chainlink service. Currently, the customer team uses the chainlink service to obtain the real-time prices of BTC, HT and ETH to calculate the random number based on the random algorithm, which increases the difficulty of inferring random number.

LLC-06 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/LuckyLuckyChef.sol: 166, 171, 176, 213, 226, 274, 350, 373	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit c5f3f012c33f7c3d2c4621f92e05916b51381d58.

LLC-07 | Redundant Data

Category	Severity	Location	Status
Logical Issue	Informational	contracts/LuckyLuckyChef.sol: 373	Declined

Description

If the user withdraws all deposits, then the user address should be removed from `userAddresses`. Although a new lottery is started, the user's power will be initialized to 0 and the user will not get rewards, but the data is redundant data and has no meaning.

Recommendation

We recommend modifying as below:

```
1       if(formerAmount == _amount){
2           delete userAddresses[msg.sender];
3       }
```

Alleviation

No alleviation.

LLC-08 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	contracts/LuckyLuckyChef.sol: 176	**Acknowledged**

Description

The owner of contract `LuckyLuckyChef` has the permission to:

1. withdraw rewardToken to owner by `withdrawRewardToken` function.

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

Customer team response:
DAO/governance/voting module will be introduced in the future.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.