Cyberscope

Audit Report

Cultiv8

May 2024

Repository https://github.com/LayerZCode/LayerZContracts

Commit 74273cb1fcd20f6197bcabee2d5c031a3da9dc44

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	08 May 2024

Source Files

Filename	SHA256
contracts/LotteryToken.sol	1119ed02c1fa45811002a6284613960815 2402153d78f37c4d3de65695dc881b
contracts/lib/PancakeAdapter.sol	99ff671b7056b01cc700dfd01e4c5c4e4b1 c83c4a9d9556ba55dc5b6631759af
contracts/lib/ConstantsAndTypes.sol	89b0d2e772aa57d7bfb304b23d3859662a b4f5fc8529326005dd977e5c9f28f3
contracts/lib/configs/VRFConsumerConfig.sol	d8610df5949ab6eb4c9aa870866b0bb023 a30dff4d116e4a6a08aa220520ecba
contracts/lib/configs/ProtocolConfig.sol	f746c779e4f51e4aaf1c80129ce3aa09f163 df974762412b66fe57363988a620
contracts/lib/configs/LotteryEngineConfig.sol	6272e15f5e21f702f5ea59f439db7a145c8f 9679472d3d8dffe34d321b871f46
contracts/lib/configs/Configuration.sol	4dcc676364bb379836d38147ee9ba05125 a328f9c598ead3f8b96e875053111f
contracts/lib/chainlink/VRFCoordinatorV2Interface.sol	d7607ab085b0ff3bc835eb4b262bc15dbfc 5a29690e11281740f94b58e02fc1d
contracts/lib/chainlink/VRFConsumerBaseV2.sol	b1871961f6844554da24b70814afba1109f 7834f02e7774e5706f8c4595443af

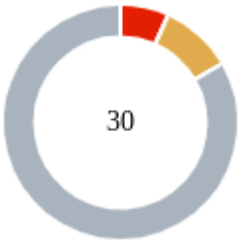
contracts/lib/chainlink/Link.sol	0f5a2069c26fcc6bb3b21984c4a2fb7ddf52d3f079948a744d69c78a69b499a2
contracts/interfaces/IPegSwap.sol	6d35b09568f477a5fe8362644bc88a812d402fdeded3e762b6ede53c52b0ede6
contracts/interfaces/IPancakeRouter02.sol	317bb97d04fc0c31fda776b93416f04fc5ddf1127e5bce27b61173a68879a6ad
contracts/interfaces/IPancakeRouter01.sol	80b3675c9e6ad87ee429cf6c71fae3b41d49929b000795890048b289ba4a2e7e
contracts/interfaces/IPancakePair.sol	5e52a76cb64aa0607e2dd207de97461d5f7ade01b469b2564e1fd0e30f39dfce
contracts/interfaces/IPancakeFactory.sol	d9490895664b99ea0c44f3608ca6b9b903e521c9ddb2c7d1aa3ac7337ed6b6bc
contracts/interfaces/IConfiguration.sol	5a14f9ce51da861408736d1fc93b680aa94d14a00c00f8d8ce8b2949bae2de80

Overview

LayerZ introduces a reflection token incorporating a lottery mechanism powered by Chainlink VRF. The lottery system comprises three distinct types:

1. **Smash Time**: Activated during buy or sell transactions exceeding $100. For every $100 increment, users receive one ticket, with a maximum of 10 tickets per transaction. Each ticket holds a 1% chance of winning, cumulating to a maximum of 10% with 10 tickets.
2. **Donation**: Participation in the donation lottery requires sending a minimum amount to the designated donation address. The quantity of donation tickets correlates with the total donation amount. Additionally, owners can generate donation tickets. This lottery type is initiated upon any transfer if sufficient eligible addresses are present.
3. **Holder**: Triggered by any transfer, the holder lottery awards tickets based on the number of transactions occurring since the last lottery event. Participants receive up to 3 tickets depending on their holding amount.

Findings Breakdown



Critical	2
Medium	3
Minor / Informative	25

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	3	0	0	0
Minor / Informative	25	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	RUA	rawFulfillRandomWords Unrestricted Access	Unresolved
●	FRWPR	Fulfilling Random Words Potential Revert	Unresolved
●	ILEH	Inadequate Lottery Exclusion Handling	Unresolved
●	IETE	Incorrect Event Transfer Emissions	Unresolved
●	CO	Code Optimization	Unresolved
●	CR	Code Repetition	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	DPI	Decimals Precision Inconsistency	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	IRTI	Improper Regular Transfer Identification	Unresolved
●	ISAC	Inconsistent Significant Amount Check	Unresolved
●	MEE	Missing Events Emission	Unresolved

●	MSC	Missing Sanity Check	Unresolved
●	PIL	Potential Inconsistent Logic	Unresolved
●	PIMTC	Potential Incorrect Max Transaction Calculation	Unresolved
●	PLR	Potential Lost Rewards	Unresolved
●	PPDLD	Potential Premature Donator Lottery Draw	Unresolved
●	RCS	Redundant Code Segments	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	TMF	Transfer Minimum Fee	Unresolved
●	UPV	Unbound Protocol Values	Unresolved
●	UAR	Unexcluded Address Restrictions	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

ST - Stops Transactions

Criticality	Critical
Location	contracts/LotteryToken.sol#L608,1526
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` to zero. As a result, the contract may operate as a honeypot.

```
if (_amount > (balanceOf(PANCAKE_PAIR) * maxBuyPercent) / PRECISION) {
    revert TransferAmountExceedsPurchaseAmount();
}
...
function setMaxBuyPercent(uint256 _maxBuyPercent) external onlyOwner {
    maxBuyPercent = _maxBuyPercent;
}
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

RUA - rawFulfillRandomWords Unrestricted Access

Criticality	Critical
Location	contracts/lib/chainlink/VRFConsumerBaseV2.sol#L127
Status	Unresolved

Description

The contract utilizes Chainlink's Verifiable Random Function (VRF) to generate random numbers, enhancing the security and unpredictability of these numbers. However, the `rawFulfillRandomWords` function, which is intended to be called only by the Chainlink VRF Coordinator, has a critical security check commented out. This check ensures that only the VRF Coordinator can invoke the function. The relevant code is shown below:

```
function rawFulfillRandomWords(uint256 requestId, uint256[] memory
randomWords) external {
    // if (msg.sender != vrfCoordinator) {
    //     revert OnlyCoordinatorCanFulfill(msg.sender, vrfCoordinator);
    // }
    fulfillRandomWords(requestId, randomWords);
}
```

In its current state, any address can call the `rawFulfillRandomWords` function, which undermines the security of the random number generation process. This lack of access control could be exploited by a malicious actor to provide arbitrary or predictable random values, leading to potential manipulation of the contract's logic that depends on these random numbers.

Recommendation

Reinstate the commented-out access control check to ensure that only the Chainlink VRF Coordinator can invoke the `rawFulfillRandomWords` function.

By implementing this check, the contract will properly restrict access to the `rawFulfillRandomWords` function, ensuring that the random values used are verifiably

provided by the trusted Chainlink VRF Coordinator, thereby maintaining the integrity and security of the contract.

FRWPR - Fulfilling Random Words Potential Revert

Criticality	Medium
Location	contracts/LotteryToken.sol#L808,822
Status	Unresolved

Description

During the audit of the contract, it was identified that the contract utilizes the Chainlink VRF (Verifiable Random Function) for randomness generation in determining lottery winners. However, a potential issue arises in the process of fulfilling random words and awarding prizes, particularly in the `_finishDonationLottery` function.

The `_finishDonationLottery` function is responsible for finalizing the donation lottery round by distributing prizes to the winners. It was observed that the function performs token swaps for BNB as part of the prize distribution process. The concern arises from the fact that the token swapping operation, specifically the `_convertDonationLotteryPrize` function, could potentially revert due to unforeseen circumstances such as insufficient liquidity or other errors in the swapping mechanism.

Furthermore, the `fulfillRandomWords` function, which is crucial for determining the randomness of lottery winners, directly calls the `_finishRound` function, initiating the prize distribution process. Consequently, if the token swapping operation within `_finishDonationLottery` reverts, it would lead to the `_finishRound` function reverting as well. This could result in the entire lottery round being aborted.

```
function _finishSmashTimeLottery(uint256 _requestId, RandomWords memory
_random) private {
    ...
    _convertSmashTimeLotteryPrize();
    ...
}
...
function _finishDonationLottery(uint256 _requestId, uint256 _random)
private {
    ...
    // convert prize
    _convertDonationLotteryPrize();
    ...
}
...
function _convertSmashTimeLotteryPrize() private {
    uint256 conversionAmount =
balanceOf(smashTimeLotteryPrizePoolAddress); //@audit function
redundancy/duplication

    _tokenTransfer(smashTimeLotteryPrizePoolAddress, address(this),
conversionAmount);

    uint256 convertedBNB = _swapTokensForBNB(conversionAmount);
    unchecked {
        smashtimeLotteryBNBPrize += convertedBNB;
    }
}
...
function _convertDonationLotteryPrize() private {
    uint256 conversionAmount =
balanceOf(donationLotteryPrizePoolAddress); //@audit function
redundancy/duplication

    _tokenTransfer(donationLotteryPrizePoolAddress, address(this),
conversionAmount);

    uint256 convertedBNB = _swapTokensForBNB(conversionAmount);
    unchecked {
        donationLotteryBNBPrize += convertedBNB;
    }
}
```

Recommendation

To mitigate the risk of reverting due to token swapping issues, the team is advised to decouple the logic responsible for fulfilling random words from the token swapping process. This separation would ensure that the fulfillment of random words and the determination of

lottery winners remain independent of any potential issues encountered during token swaps.

By implementing these recommendations, the contract can enhance its resilience against potential reverts and ensure a more reliable and secure lottery experience for participants.

ILEH - Inadequate Lottery Exclusion Handling

Criticality	Medium
Location	contracts/LotteryToken.sol#L774
Status	Unresolved

Description

The contract includes functionality to check if an address is eligible for the holder's lottery within the `_checkForHoldersLotteryEligibilities` function. However, an issue arises in the exclusion mechanism. When an address is excluded from rewards using the `excludeFromReward` function, it is correctly prevented from participating in future lottery rounds. However, the oversight occurs when an address, which was previously eligible for the lottery, gets excluded from rewards later on. In such cases, the address remains eligible for the lottery indefinitely, despite being excluded from rewards.

This vulnerability arises due to the failure to update the lottery eligibility status of addresses dynamically. The `_checkForHoldersLotteryEligibility` function determines the eligibility status of addresses based on their balance and exclusion status at the time of the function call. Once an address becomes eligible for the lottery, its eligibility status remains unchanged, even if it is subsequently excluded from rewards.

```
function _checkForHoldersLotteryEligibilities(address _transferrer,
address _recipient, uint256 _amount) private {
    // exit if holders lottery is disabled
    if (!_lotteryConfig.holdersLotteryEnabled) {
        return;
    }

    /*
increment counter, if current transfer amount worth more than 10$,
and recipient or transferrer initiated the transaction.
    */

    if (_isSignificant(_amount, significantAmount) && (tx.origin ==
_recipient || tx.origin == _transferrer)) {
        _holdersLotteryTxCounter++;
    }

    // read current treshold
    uint256 threshold =
_holdersEligibilityThreshold(_lotteryConfig.holdersLotteryMinPercent);

    // check transferrer and recipient for eligibility
    _checkForHoldersLotteryEligibility(_transferrer, threshold);
    _checkForHoldersLotteryEligibility(_recipient, threshold);

    // if conditions are met, trigger the lottery
    if (
        _lotteryConfig.holdersLotteryEnabled && _holdersLotteryTxCounter
>= _lotteryConfig.holdersLotteryTxTrigger
            && _holders.first.length != 0
    ) {
        _triggerHoldersLottery();
    }
}
```

Recommendation

The team is advised to address the identified vulnerability in the lottery system, it is crucial to implement a dynamic update mechanism for the eligibility status of addresses. This entails ensuring that addresses excluded from rewards are promptly disqualified from participating in the lottery.

IETE - Incorrect Event Transfer Emissions

Criticality	Medium
Location	contracts/LotteryToken.sol#L383
Status	Unresolved

Description

The emissions of `Transfer` events within `_reflectFee` function are incorrect. The events are being emitted with `msg.sender` as the `from` address, which will not accurately reflect the actual token owner in the case where `transferFrom` function calls `_reflectFee`. This is because `msg.sender` in the context of `transferFrom` is the approved spender, not the token owner.

This issue causes misleading `Transfer` event logs, where the `from` address shows the spender instead of the actual owner. This can lead to incorrect assumptions and tracking errors in off-chain systems that rely on these events for token ownership records and history.

```
function _reflectFee(RInfo memory _rr, TInfo memory _tt) private {
    ...
    // emit transfers for each fee distribution
    emit Transfer(msg.sender, holderLotteryPrizePoolAddress,
_tt.tHolderPrizeFee);
    emit Transfer(msg.sender, smashTimeLotteryPrizePoolAddress,
_tt.tSmashTimePrizeFee);
    emit Transfer(msg.sender, teamFeesAccumulationAddress,
_tt.tDevFundFee);
    emit Transfer(msg.sender, treasuryFeesAccumulationAddress,
_tt.tTreasuryFee);
    emit Transfer(msg.sender, donationLotteryPrizePoolAddress,
_tt.tDonationLotteryPrizeFee);
    emit Transfer(msg.sender, DEAD_ADDRESS, _tt.tBurnFee);
}
```

Recommendation

The team is advised to ensure the `Transfer` events reflect the correct owner, the actual `from` address should be passed as a parameter to the `_reflectFee` function.

Implementing the above recommendation will ensure that the `Transfer` events correctly reflect the original token owner's address, thereby maintaining accurate and reliable event logs for tracking token movements and ownership changes.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1310,1378
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The function `_addDonationsLotteryTickets` contains a redundant conditional check towards the end of the function to verify if donations lottery is enabled and if the number of unique donators exceeds the minimum required for participation. However, this check is unnecessary since the function already begins by verifying if the donations lottery is enabled.

Furthermore, the `_uniqueDonatorsCounter` could be checked only after the variable has been incremented.

Finally, the `_uniqueDonatorsCounter` is updated within a for loop, that could be optimized by using a local variable and assigning it to the `_uniqueDonatorsCounter` variable after the end of the for loop.

```
function _addDonationsLotteryTickets(address _transferrer, address
_recipient, uint256 _amount) private {
    if (!_lotteryConfig.donationsLotteryEnabled) {
        return;
    }
    // if this transfer is a donation, add a ticket for transferrer.
    if (_recipient == _lotteryConfig.donationAddress && _amount >=
_lotteryConfig.minimalDonation) {
        if (_donatorTicketIdxs[_donationRound][_transferrer].length ==
0) {
            _uniqueDonatorsCounter++;
        }
        uint256 length = _donators.length;
        _donators.push(_transferrer);
        _donatorTicketIdxs[_donationRound][_transferrer].push(length);
    }
    if (_lotteryConfig.donationsLotteryEnabled && _uniqueDonatorsCounter
>= _lotteryConfig.minimumDonationEntries) {
        _donationsLottery();
    }
}
...
function mintDonationTickets(address[] calldata _recipients, uint256[]
calldata _amounts) external onlyOwner {
    uint256 recipientsLength = _recipients.length;
    if (recipientsLength != _amounts.length) {
        revert RecipientsLengthNotEqualToAmounts();
    }

    uint256 round = _donationRound;
    for (uint256 i = 0; i < recipientsLength;) {
        address recipient = _recipients[i];
        uint256 amount = _amounts[i];
        uint256 idx = _donatorTicketIdxs[round][recipient].length;
        uint256 newIdx = idx + amount;

        if (_donatorTicketIdxs[round][recipient].length == 0) {
            _uniqueDonatorsCounter++;
        }

        for (; idx < newIdx;) {
            _donators.push(recipient);
            _donatorTicketIdxs[round][recipient].push(idx);

            unchecked {
                ++idx;
            }
        }

        unchecked {
```

```
                ++i;
            }
        }
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L808,822
Status	Unresolved

Description

During the assessment, we identified redundant logic in the contract related to prize conversion for two different lotteries, namely the SmashTime Lottery and the Donation Lottery. Specifically, the contract contains two private methods named `_convertSmashTimeLotteryPrize` and `_convertDonationLotteryPrize`, both of which essentially perform the same operations with minor differences in variable names.

In both methods, the following steps are executed:

1. Determine the amount of tokens held in the respective lottery prize pool.
2. Transfer these tokens to the contract.
3. Convert the transferred tokens into BNB (Binance Coin) using the `_swapTokensForBNB` function.
4. Update the BNB prize balance for the corresponding lottery.

While the functionality is identical, the only discrepancies lie in the variables used to identify the prize pool address and the BNB prize balance storage variable.

```
function _convertSmashTimeLotteryPrize() private {
    uint256 conversionAmount =
balanceOf(smashTimeLotteryPrizePoolAddress);

    _tokenTransfer(smashTimeLotteryPrizePoolAddress, address(this),
conversionAmount);

    uint256 convertedBNB = _swapTokensForBNB(conversionAmount);
    unchecked {
        smashtimeLotteryBNBPrize += convertedBNB;
    }
}

/**
    * Convert prize for Donation Lottery.
    */
function _convertDonationLotteryPrize() private {
    uint256 conversionAmount =
balanceOf(donationLotteryPrizePoolAddress);

    _tokenTransfer(donationLotteryPrizePoolAddress, address(this),
conversionAmount);

    uint256 convertedBNB = _swapTokensForBNB(conversionAmount);
    unchecked {
        donationLotteryBNBPrize += convertedBNB;
    }
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1365,1470,1487,1507,1522,1526,1530,1534,1538,1542,1547,1551 contracts/lib/Configuration.sol#L53,59,63,67,73,77,83,89,95,99,103,107,111,115,119,123,127,131,135,139,143,147,151,155,159,168,172,176,185
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function mintDonationTickets(address[] calldata _recipients, uint256[]
calldata _amounts) external onlyOwner
...
function updateHolderList(address[] calldata _holdersToCheck) external
onlyOwner
...
function updateHolderList(address[] calldata _holdersToCheck) external
onlyOwner
...
function includeInReward(address _account) external onlyOwner
...
function setWhitelist(address _account, bool _status) external onlyOwner
...
function setMaxBuyPercent(uint256 _maxBuyPercent) external onlyOwner
...
function setSwapAndLiquifyEnabled(bool _enabled) external onlyOwner
...
function setLiquiditySupplyThreshold(uint256 _amount) external onlyOwner
...
function setFeeSupplyThreshold(uint256 _amount) external onlyOwner
...
function setThreeDaysProtection(bool _enabled) external onlyOwner
...
function withdraw(uint256 _amount) external onlyOwner
...
function withdrawBNB(uint256 _amount) external onlyOwner
...
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1283
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
uint256 usdAmount = _TokenPriceInUSD(_amount) / 1e18;
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals

Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/lib/PancakeAdapter.sol#L20,21
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_WBNB_ADDRESS
_USDT_ADDRESS
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

IRTI - Improper Regular Transfer Identification

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L906,456
Status	Unresolved

Description

The contract contains a logical error in the `_tokenTransfer` function where it determines whether a transfer is a regular transfer (i.e., not a buy or sell transaction). Specifically, the contract incorrectly uses the OR (`||`) operator instead of the AND (`&&`) operator when checking if neither the sender nor the recipient is the PancakeSwap pair address (`PANCAKE_PAIR`). This logical error results in misclassification of transfer types, potentially leading to incorrect fee application and transfer handling.

```
function _tokenTransfer(address _sender, address _recipient, uint256
_amount) private {
    ...
    // if pair is not involved => its a regular transfer
    bool regularTransfer = _sender != PANCAKE_PAIR || _recipient !=
PANCAKE_PAIR;
    ...
}
...
if (_regularTransfer) {
    _fee = _fee > ONE_PERCENT ? _fee : ONE_PERCENT;
}
```

Recommendation

The team is advised to review and update the contract logic to ensure that regular transfers are correctly identified.

ISAC - Inconsistent Significant Amount Check

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L774 contracts/lib/configs/ProtocolConfig.sol#L24,73
Status	Unresolved

Description

The contract includes a feature where certain actions, such as incrementing a counter, are occuring when the transfer amount is "significant." According to the comments, this significance is defined as a transfer amount worth more than $10. However, in the actual implementation, the contract uses a variable `significantAmount` to determine if a transfer is significant. This variable is set to `1e19` (10^19) by default but can be modified through a setter function `_setSignificantAmountForTransfer` .

The description and the code implementation are inconsistent. The documentation specifies a hardcoded value of $10, but the actual check uses a variable that can be altered. This discrepancy can lead to unintended behavior:

```
function _checkForHoldersLotteryEligibilities(address _transferrer,
address _recipient, uint256 _amount) private {
    ...
    if (_isSignificant(_amount, significantAmount) && (tx.origin ==
_recipient || tx.origin == _transferrer)) {
        _holdersLotteryTxCounter++;
    }
    ...
}
...
uint256 public significantAmount = 1e19;
...
function _setSignificantAmountForTransfer(uint256 _value) internal {
    significantAmount = _value;
}
```

Recommendation

The comments should be clarified to reflect that the `significantAmount` is a variable that can be adjusted and provide guidance on its appropriate usage and limits. Additionally,

validation checks should be implemented within the setter method to ensure that the new value set for `significantAmount` is reasonable and within expected bounds, thereby maintaining the integrity and intended functionality of the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L555,1499,1516,1522,1526,1530,1534,1538,1542 contracts/lib/configs/Configuration.sol
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function _takeLiquidity(uint256 _rLiquidity, uint256 _tLiquidity)
private
...
function excludeFromReward(address _account) public onlyOwner
...
function includeInReward(address _account) external onlyOwner
...
function setWhitelist(address _account, bool _status) external onlyOwner
...
function setMaxBuyPercent(uint256 _maxBuyPercent) external onlyOwner
...
function setSwapAndLiquifyEnabled(bool _enabled) external onlyOwner
...
function setLiquiditySupplyThreshold(uint256 _amount) external onlyOwner
...
function setFeeSupplyThreshold(uint256 _amount) external onlyOwner
...
function setThreeDaysProtection(bool _enabled) external onlyOwner
...
function setHolderLotteryPrizePoolAddress(
    address _newAddress
) external onlyOwner
...
function setSmashTimeLotteryPrizePoolAddress(
    address _newAddress
) external onlyOwner
...
function setDonationLotteryPrizePoolAddress(
    address _newAddress
) external onlyOwner
...
function setTeamAddress(address _newAddress) external onlyOwner
...
function setTeamAccumulationAddress(address _newAddress) external
onlyOwner
...
function setTreasuryAddress(address _newAddress) external onlyOwner
...
function setTreasuryAccumulationAddress(address _newAddress) external
onlyOwner
...
function setFeeConfig(uint256 _feeConfigRaw) external onlyOwner
...
function switchSmashTimeLotteryFlag(bool flag) external onlyOwner
...
function switchHoldersLotteryFlag(bool flag) external onlyOwner
...
function switchDonationsLotteryFlag(bool flag) external onlyOwner
...
function excludeFromFee(address account) external onlyOwner
```

```
...
function includeInFee(address account) external onlyOwner
...
function setHoldersLotteryTxTrigger(uint64 _txAmount) external onlyOwner
...
function setHoldersLotteryMinPercent(uint256 _minPercent) external
onlyOwner
...
function setDonationAddress(address _donationAddress) external onlyOwner
...
function setMinimalDonation(uint256 _minimalDonation) external onlyOwner
...
function setFee(uint256 _fee) external onlyOwner
...
function setMinimumDonationEntries(uint64 _minimumEntries) external
onlyOwner
...
function setSmashTimePrizePercent(uint256 _value) external onlyOwner
...
function setHoldersLottreryPrizePercent(uint256 _value) external
onlyOwner
...
function setDonationLotteryPrizePercent(uint256 _value) external
onlyOwner
...
function setTreasuryPlainTokenPercent(uint256 _value) external onlyOwner
...
function setSignificantAmountForTransfer(uint256 _value) external
onlyOwner
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L187,1526,1534,1538 contracts/lib/Configuration.sol#L53
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The function arguments are not properly sanitized.

```
constructor(
    address _mintSupplyTo,
    address _coordinatorAddress,
    address _routerAddress,
    address _wbnbAddress,
    address _tusdAddress,
    uint256 _fee,
    ConsumerConfig memory _consumerConfig,
    DistributionConfig memory _distributionConfig,
    LotteryConfig memory _lotteryConfig
)
    VRFConsumerBaseV2(_coordinatorAddress)
    PancakeAdapter(_routerAddress, _wbnbAddress, _tusdAddress)
    Configuration(_fee, _consumerConfig, _distributionConfig,
_lotteryConfig)
{
...
function setMaxBuyPercent(uint256 _maxBuyPercent) external onlyOwner
...
function setLiquiditySupplyThreshold(uint256 _amount) external onlyOwner
...
function setFeeSupplyThreshold(uint256 _amount) external onlyOwner
...
function setFeeConfig(uint256 _feeConfigRaw) external onlyOwner
...
function setFee(uint256 _fee) external onlyOwner
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PIL - Potential Inconsistent Logic

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L577
Status	Unresolved

Description

During the assessment, we identified a potential inconsistency in the logic of the `_antiAbuse` function, particularly in the condition checking for the allowed transfer amount compared to the user's balance.

The `_antiAbuse` function is designed to prevent abuse, particularly targeting bot or whale activities by imposing daily limits on transfers. However, we observed that the condition for reverting the transaction due to the transfer amount exceeding the daily limit may not be implemented as intended.

In the function, after calculating `lastUserBalance` by adding the current transfer amount to the recipient's balance, the code checks if the `lastUserBalance` is greater than or equal to the `allowedAmount` based on the daily limit. If this condition evaluates to true, the function reverts the transaction with the message `TransferAmountExceededForToday`.

However, potentially the transaction should revert only if the `lastUserBalance` exceeds the `allowedAmount`, indicating that the transfer would exceed the daily limit.

```
    function _antiAbuse(address _from, address _to, uint256 _amount)
private view {
        // If owner, skip checks
        if (_from == owner() || _to == owner()) return;

        (, uint256 tSupply) = _getCurrentSupply();

        // read user balance and add current transfer amount.
        uint256 lastUserBalance = balanceOf(_to) + ((_amount *
(PRECISION - _calcFeePercent())) / PRECISION);

        // Bot / whales prevention
        if (threeDaysProtectionEnabled) {
            uint256 timeSinceCreation = block.timestamp -
_CREATION_TIME;
            uint256 dayLimit = 0;

            if (timeSinceCreation <= 1 days) {
                dayLimit = DAY_ONE_LIMIT;
            } else if (timeSinceCreation <= 2 days) {
                dayLimit = DAY_TWO_LIMIT;
            } else if (timeSinceCreation <= 3 days) {
                dayLimit = DAY_THREE_LIMIT;
            }

            if (dayLimit > 0) {
                uint256 allowedAmount = (tSupply * dayLimit) /
PRECISION;
                if (lastUserBalance >= allowedAmount) {
                    revert TransferAmountExceededForToday();
                }
            }
        }

        // cant exceed tx max buy percent
        if (_amount > (balanceOf(PANCAKE_PAIR) * maxBuyPercent) /
PRECISION) {
            revert TransferAmountExceedsPurchaseAmount();
        }
    }
```

Recommendation

To address this inconsistency and ensure that the function behaves as intended, the team is advised to revise the condition to the intended behavior.

PIMTC - Potential Incorrect Max Transaction Calculation

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L584,906
Status	Unresolved

Description

The contract includes a mechanism to enforce a maximum transaction limit, ensuring that no single transfer exceeds a predefined percentage of the total supply. The contract also calculates and imposes fees on token transfers, which are intended to be factored into the max transaction limit check. However, the implementation incorrectly assumes the fee will be applied to the current transaction, potentially leading to inaccurate validation.

In the `_antiAbuse` function, the contract attempts to prevent large transactions by comparing the transaction amount against a percentage of the total supply. The relevant code snippet is:

```
uint256 lastUserBalance = balanceOf(_to) + ((_amount * (PRECISION -
_calcFeePercent())) / PRECISION);
...
uint256 allowedAmount = (tSupply * dayLimit) / PRECISION;
if (lastUserBalance >= allowedAmount) {
    revert TransferAmountExceededForToday(); //@audit lastUserBalance =
allowed amount should not revert
}
```

Here, the `_amount` is adjusted by subtracting the calculated fee percentage (`_calcFeePercent()`). This adjusted amount is then compared against the allowable max transaction limit. However, the fee might not be applicable to the current transaction due to various conditions (e.g., addresses being excluded from fees), leading to a discrepancy between the actual transaction amount and the assumed amount used in the validation.

As a result, transactions that should be permitted might be incorrectly reverted, and transactions that should be restricted might bypass the limitation.

Recommendation

The team is advised to refactor the `_antiAbuse` function to correctly account for the actual transaction amount after fees are applied, ensuring accurate enforcement of the max transaction limit. This can be achieved by determining the fee applicability before performing the max transaction limit check.

This adjustment ensures that the actual amount being transferred (after fees) is used to check against the max transaction limit, thereby aligning the logic with the intended behavior.

PLR - Potential Lost Rewards

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1105,1215
Status	Unresolved

Description

During the assessment a potential issue was identified. The contract utilizes the Chainlink Verifiable Random Function (VRF) to determine lottery winners, and upon receiving random words from the Chainlink VRF, the function `fulfillRandomWords` is called. This function subsequently invokes the `_finishRound` function, which then calls the appropriate lottery finish function based on the lottery type.

Specifically, in the `_finishSmashTimeLottery` and `_finishDonationLottery` functions, there are instances where native token transfers occur to distribute prizes to lottery winners. However, these transfers are implemented without error handling since the fulfillRandomWords function must not revert. As a consequence, the winners may not receive their rewards, and there is no mechanism in place for them to claim their prizes later.

```
function _finishSmashTimeLottery(uint256 _requestId, RandomWords memory
_random) private {
    // read lottery info
    LotteryRound storage _round = rounds[_requestId];
    address player = _round.player;

    // create array of 100 empty addresses.
    address[100] memory tickets;

    // seed array with player address equally to amount of entries
    for (uint256 i = 0; i < uint8(_round.entry);) {
        uint256 shift = (i * TWENTY_FIVE_BITS);
        // indices are chosen with second random word
        uint256 idx = (_random.second >> shift);
        assembly {
            idx := mod(idx, 100)
        }
        _seedTicketsArray(tickets, idx, player);
        unchecked {
            ++i;
        }
    }

    // select winner idx with first random word
    uint256 winnerIdx;
    assembly {
        winnerIdx := mod(mload(_random), 100)
    }
    _convertSmashTimeLotteryPrize();
    // if selected ticket is a winning one, convert and pay out the
prize.
    if (tickets[winnerIdx] == player) {
        uint256 prize = _calculateSmashTimeLotteryPrize();

        // chainlnk doc states, that we should not revert callback txs.
        assembly {
            function transfer_unsafe(recipient, amount) {
                pop(call(gas(), recipient, amount, 0, 0, 0, 0))
            }
            transfer_unsafe(player, prize)
        }

        unchecked {
            smashtimeLotteryBNBPrize -= prize;
        }

        emit SmashTimeLottery(_requestId, player, prize);
    }

    _round.lotteryType = LotteryType.FINISHED_SMASHTIME;
```

```
}
```

Recommendation

The team is recommended to revise the logic to safely distribute the rewards to the winners.

PPDLD - Potential Premature Donator Lottery Draw

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1342
Status	Unresolved

Description

In the function `transferDonationTicket`, there is an oversight regarding the management of `_uniqueDonatorsCounter`. According to the current implementation, when a donation ticket is transferred from one address to another, the `_uniqueDonatorsCounter` is only incremented if the receiving address has zero donation tickets left. However, there is no corresponding check to decrement `_uniqueDonatorsCounter` when the transferring address has zero donation tickets left after the transfer.

In the scenario where `length == 1`, indicating that the transferring address has only one donation ticket left, the code should include logic to decrement `_uniqueDonatorsCounter` because after this transfer, the transferring address will have zero donations left. Without this check, the `_uniqueDonatorsCounter` may become inaccurate, potentially leading to premature lottery draws.

```
function transferDonationTicket(address _to) external {
    // get current Donation lottery round
    uint256 round = _donationRound;

    // read length of donator tickets and revert if there is nothing to
transfer
    uint256 length = _donatorTicketIdxs[round][msg.sender].length;
    if (length == 0) {
        revert NoDonationTicketsToTransfer();
    }

    // transfer ticket
    uint256 idx = _donatorTicketIdxs[round][msg.sender][length - 1];
    _donatorTicketIdxs[round][msg.sender].pop();
    _donators[idx] = _to;
    if (_donatorTicketIdxs[round][_to].length == 0) {
        _uniqueDonatorsCounter++;
    }
    _donatorTicketIdxs[round][_to].push(idx);
}
```

Recommendation

The team is advised to address this issue, by introducing the appropriate checks changes to the `_uniqueDonatorsCounter` protocol variable.

RCS - Redundant Code Segments

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L138,139,145,246,253
Status	Unresolved

Description

The contract is currently containing code segments, that do not provide any actual functionality. Such redundant code segments can lead to confusion and misinterpretation of the contract's purpose and functionality. Moreover, they contribute to unnecessary bloat in the contract, potentially impacting its efficiency and clarity.

```
// TODO:  use real value
```

Recommendation

It is recommended to remove these redundant code segments from the contract. Eliminating these non-functional parts will streamline the contract, making it more efficient and easier to comprehend. This action will also reduce the potential for confusion among users and developers who interact with or audit the contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1522,1530,1542
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function setWhitelist(address _account, bool _status) external onlyOwner
{
    whitelist[_account] = _status;
}
...
function setSwapAndLiquifyEnabled(bool _enabled) external onlyOwner {
    swapAndLiquifyEnabled = _enabled;
}
...
function setThreeDaysProtection(bool _enabled) external onlyOwner {
    threeDaysProtectionEnabled = _enabled;
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L664,697,698
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
function _distributeFees() private lockTheSwap {
    _distributeFeeToAddress(teamFeesAccumulationAddress, teamAddress);
    _distributeFeeToAddress(treasuryFeesAccumulationAddress,
treasuryAddress);
}
...
function _distributeFeeToAddress(address _feeAccumulationAddress,
address _destinationAddress) private {
    ...
    _swapTokensForTUSDT(half, _destinationAddress);
    _swapTokensForBNB(accumulatedBalance - half, _destinationAddress);
    ...
}
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

TMF - Transfer Minimum Fee

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L441
Status	Unresolved

Description

The function `_getTValues` is responsible for calculating various transaction values, including fees, based on the amount being transferred (`_tAmount`), whether fees should be taken (`_takeFee`), and whether the transfer is regular (`_regularTransfer`).

The function checks `_fee = _fee > ONE_PERCENT ? _fee : ONE_PERCENT;` suggesting that the fee should be set to at least 1%. This fee is taken only on regular transfers,

This means that even if `_fee` is set to 0 or any value less than 1%, the function imposes a minimum fee of 1% for regular transfers.

```
function _getTValues(uint256 _tAmount, bool _takeFee, bool
_regularTransfer)
    private
    view
    returns (TInfo memory tt)
{
    // if no fees taken, all token should be transferred. //@audit-ok
    if (!_takeFee) {
        tt.tTransferAmount = _tAmount;
        return tt;
    }

    // get fees
    uint256 _fee = _calcFeePercent();

    // tax transfers from one EOA to another, even if fees are 0%.
    if (_regularTransfer) {
        _fee = _fee > ONE_PERCENT ? _fee : ONE_PERCENT; //@audit
TRANSFERS HAVE 1% FEES MINIMUM
    }

    // read fees
    Fee fees = _fees;

    // Combined calculation for efficiency
    unchecked {
        tt.tBurnFee = (fees.burnFeePercent(_fee) * _tAmount) /
PRECISION;
        tt.tDistributionFee = (fees.distributionFeePercent(_fee) *
_tAmount) / PRECISION;
        tt.tTreasuryFee = (fees.treasuryFeePercent(_fee) * _tAmount) /
PRECISION;
        tt.tDevFundFee = (fees.devFeePercent(_fee) * _tAmount) /
PRECISION;
        tt.tSmashTimePrizeFee =
(fees.smashTimeLotteryPrizeFeePercent(_fee) * _tAmount) / PRECISION;
        tt.tHolderPrizeFee = (fees.holdersLotteryPrizeFeePercent(_fee) *
_tAmount) / PRECISION;
        tt.tDonationLotteryPrizeFee =
(fees.donationLotteryPrizeFeePercent(_fee) * _tAmount) / PRECISION;
        tt.tLiquidityFee = (fees.liquidityFeePercent(_fee) * _tAmount) /
PRECISION;

        uint256 totalFee = tt.tBurnFee + tt.tLiquidityFee +
tt.tDistributionFee + tt.tTreasuryFee + tt.tDevFundFee
            + tt.tSmashTimePrizeFee + tt.tDonationLotteryPrizeFee +
tt.tHolderPrizeFee;

        tt.tTransferAmount = _tAmount - totalFee;
    }
```

```
    return tt;
}
```

Recommendation

To address this finding, team is advised to consider reviewing and potentially revising the logic within the `_getTValues` function to ensure that the fee calculation for regular transfers respects the value of `_fee` as intended.

UPV - Unbound Protocol Values

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1526,1534,1538 contracts/lib/configs/ProtocolConfig.sol#L65,69,73
Status	Unresolved

Description

The contract contains multiple setter functions that allow the owner to update important parameters. Specifically, the functions `setMaxBuyPercent`, `setLiquiditySupplyThreshold`, and `setFeeSupplyThreshold` do not enforce any upper or lower bounds on the values that can be set.

```
function setMaxBuyPercent(uint256 _maxBuyPercent) external onlyOwner {
    maxBuyPercent = _maxBuyPercent;
}

function setLiquiditySupplyThreshold(uint256 _amount) external onlyOwner
{
    liquiditySupplyThreshold = _amount;
}

function setFeeSupplyThreshold(uint256 _amount) external onlyOwner {
    feeSupplyThreshold = _amount;
}
...
function _setFeeConfig(uint256 _feeConfigRaw) internal {
    _fees = Fee.wrap(_feeConfigRaw);
}

function _setTreasuryPlainTokenPercent(uint256 _value) internal {
    plainTokenPercent = _value;
}

function _setSignificantAmountForTransfer(uint256 _value) internal {
    significantAmount = _value;
}
```

Recommendation

To mitigate these risks, the team is advised to introduce reasonable upper and lower bounds for each of these parameters. This can be done by adding validation logic within each setter function. By implementing these bounds, the contract can ensure that these critical parameters remain within a safe and reasonable range, thereby reducing the risk of misconfiguration and enhancing the overall security and reliability of the contract.

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L456
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```
if (_regularTransfer) {
    _fee = _fee > ONE_PERCENT ? _fee : ONE_PERCENT;
}
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L140
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private _tTotal = 20_000_000_000 * 1e18
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L163,267,281,292,302,321,344,354,366,879,1139,1237,1342,1365,1416,1425,1434,1470,1487,1507,1522,1526,1530,1534,1538,1542,1547,1551 contracts/lib/PancakeAdapter.sol#L8,9,11,13,103,104 contracts/lib/ConstantsAndTypes.sol#L109,125,145,215,238,245,266,267,272,279,300,301,307,308,317,318,324,325 contracts/lib/configs/Configuration.sol#L23,54,59,63,68,73,78,84,90,95,99,103,107,111,135,139,143,147,151,155,159,168,172,176,185 contracts/interfaces/IPancakeRouter01.sol#L8 contracts/interfaces/IPancakeFactory.sol#L24
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
VRFCoordinatorV2Interface private _COORDINATOR
address _account
address _recipient
uint256 _amount
address _spender
address _owner
address _sender
uint256 _addedValue
uint256 _subtractedValue
uint256 _rAmount
uint256[] memory _randomWords
uint256 _requestId
recipient
amount

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L1527,1539
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
maxBuyPercent = _maxBuyPercent
feeSupplyThreshold = _amount
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/lib/ConstantsAndTypes.sol#L118,119,120,121,135,137,138,139,140,141
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
prize = (accumulated * uint32(val)) / PRECISION
first = (prize * uint32(val >> 32)) / PRECISION
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L194,196,292,569 contracts/lib/configs/Configuration.sol#L28,30
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
ConsumerConfig memory _consumerConfig
LotteryConfig memory _lotteryConfig
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/LotteryToken.sol#L868,1118,1139,1236,1237
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            _words := add(_array, ONE_WORD)
        }

assembly {
                idx := mod(idx, 100)
            }

function transfer_unsafe(recipient, amount) {
                    pop(call(gas(), recipient, amount, 0, 0, 0, 0))
                }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TestZ	Implementation	VRFConsumerBaseV2, PancakeAdapter, Configuration		
		Public	✓	VRFConsumerBaseV2 PancakeAdapter Configuration
	name	External		-
	symbol	External		-
	totalSupply	External		-
	balanceOf	Public		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	tokenFromReflection	Public		-
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		

	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	_approve	Private	✓	
	_antiAbuse	Private		
	_transfer	Private	✓	swapLockOnPairCall
	_distributeFees	Private	✓	lockTheSwap
	_distributeFeeToAddress	Private	✓	
	_checkForHoldersLotteryEligibility	Private	✓	
	_holdersEligibilityThreshold	Private		
	_checkForHoldersLotteryEligibilities	Private	✓	
	_convertSmashTimeLotteryPrize	Private	✓	
	_convertDonationLotteryPrize	Private	✓	
	_lotteryOnTransfer	Private	✓	
	_requestRandomWords	Private	✓	
	_toRandomWords	Private		
	fulfillRandomWords	Internal	✓	
	_swapAndLiquify	Private	✓	lockTheSwap
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	

	_transferFromExcluded	Private	✓	
	_transferBothExcluded	Private	✓	
	totalFeePercent	External		-
	_finishRound	Private	✓	
	_calculateSmashTimeLotteryPrize	Private		
	_seedTicketsArray	Private		
	_finishSmashTimeLottery	Private	✓	
	_finishHoldersLottery	Private	✓	
	_finishDonationLottery	Private	✓	
	_smashTimeLottery	Private	✓	
	_triggerHoldersLottery	Private	✓	
	_addDonationsLotteryTickets	Private	✓	
	_donationsLottery	Private	✓	
	transferDonationTicket	External	✓	-
	mintDonationTickets	External	✓	onlyOwner
	holdersLotteryTickets	External		-
	donationLotteryTickets	External		-
	donationLotteryTicketsAmountPerDonator	External		-
	donate	External	✓	-
	availableHoldersLotteryTickets	External		-
	claimHoldersLotteryTickets	External	✓	-
	updateHolderList	External	✓	onlyOwner
	excludeFromReward	Public	✓	onlyOwner

	includeInReward	External	✓	onlyOwner
	setWhitelist	External	✓	onlyOwner
	setMaxBuyPercent	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	External	✓	onlyOwner
	setLiquiditySupplyThreshold	External	✓	onlyOwner
	setFeeSupplyThreshold	External	✓	onlyOwner
	setThreeDaysProtection	External	✓	onlyOwner
	withdraw	External	✓	onlyOwner
	withdrawBNB	External	✓	onlyOwner
TypesHelpers	Library			
	getPrizes	Internal		
	getPrizes	Internal		
	compact	Internal		
	burnFeePercent	Internal		
	liquidityFeePercent	Internal		
	distributionFeePercent	Internal		
	treasuryFeePercent	Internal		
	devFeePercent	Internal		
	smashTimeLotteryPrizeFeePercent	Internal		
	holdersLotteryPrizeFeePercent	Internal		
	donationLotteryPrizeFeePercent	Internal		
	allTickets	Internal		

	addFirst	Internal	✓	
	removeFirst	Internal	✓	
	existsFirst	Internal		
	addSecond	Internal	✓	
	removeSecond	Internal	✓	
	existsSecond	Internal		
	addThird	Internal	✓	
	existsThird	Internal		
	removeThird	Internal	✓	
VRFConsumerConfig	Implementation			
		Public	✓	-
	_setConfig	Internal	✓	
	_setSubscriptionId	Internal	✓	
	_setCallbackGasLimit	Internal	✓	
	_setRequestConfirmations	Internal	✓	
	_setGasPriceKey	Internal	✓	
ProtocolConfig	Implementation			
		Public	✓	-
	_setHolderLotteryPrizePoolAddress	Internal	✓	
	_setSmashTimeLotteryPrizePoolAddress	Internal	✓	
	_setDonationLotteryPrizePoolAddress	Internal	✓	

	_setTeamAddress	Internal	✓	
	_setTeamAccumulationAddress	Internal	✓	
	_setTreasuryAccumulationAddress	Internal	✓	
	_setTreasuryAddress	Internal	✓	
	_setFeeConfig	Internal	✓	
	_setTreasuryPlainTokenPercent	Internal	✓	
	_setSignificantAmountForTransfer	Internal	✓	
LotteryEngineConfig	Implementation			
		Public	✓	-
	_setSmashTimePrizePercent	Internal	✓	
	_setHoldersLottreryPrizePercent	Internal	✓	
	_setDonationLotteryPrizePercent	Internal	✓	
	_switchSmashTimeLotteryFlag	Internal	✓	
	_switchHoldersLotteryFlag	Internal	✓	
	_setHoldersLotteryTxTrigger	Internal	✓	
	_setHoldersLotteryMinPercent	Internal	✓	
	_setDonationAddress	Internal	✓	
	_switchDonationsLotteryFlag	Internal	✓	
	_setMinimanDonation	Internal	✓	
	_setMinimumDonationEntries	Internal	✓	

Configuration	Implementation	IConfiguration, VRFConsumerConfig, ProtocolConfig, LotteryEngineConfig, Ownable		
		Public	✓	VRFConsumerConfig ProtocolConfig LotteryEngineConfig
	_calcFeePercent	Internal		
	setConsumerConfig	External	✓	onlyOwner
	setSubscriptionId	External	✓	onlyOwner
	setCallbackGasLimit	External	✓	onlyOwner
	setRequestConfirmations	External	✓	onlyOwner
	setGasPriceKey	External	✓	onlyOwner
	setHolderLotteryPrizePoolAddress	External	✓	onlyOwner
	setSmashTimeLotteryPrizePoolAddress	External	✓	onlyOwner
	setDonationLotteryPrizePoolAddress	External	✓	onlyOwner
	setTeamAddress	External	✓	onlyOwner
	setTeamAccumulationAddress	External	✓	onlyOwner
	setTreasuryAddress	External	✓	onlyOwner
	setTreasuryAccumulationAddress	External	✓	onlyOwner
	setFeeConfig	External	✓	onlyOwner
	switchSmashTimeLotteryFlag	External	✓	onlyOwner
	switchHoldersLotteryFlag	External	✓	onlyOwner
	switchDonationsLotteryFlag	External	✓	onlyOwner

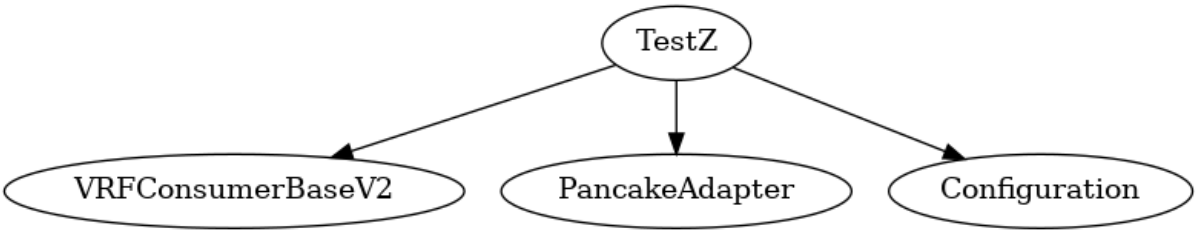
	excludeFromFee	External	✓	onlyOwner
	includeInFee	External	✓	onlyOwner
	setHoldersLotteryTxTrigger	External	✓	onlyOwner
	setHoldersLotteryMinPercent	External	✓	onlyOwner
	setDonationAddress	External	✓	onlyOwner
	setMinimalDonation	External	✓	onlyOwner
	setFee	External	✓	onlyOwner
	setMinimumDonationEntries	External	✓	onlyOwner
	setSmashTimePrizePercent	External	✓	onlyOwner
	setHoldersLottreryPrizePercent	External	✓	onlyOwner
	setDonationLotteryPrizePercent	External	✓	onlyOwner
	setTreasuryPlainTokenPercent	External	✓	onlyOwner
	setSignificantAmountForTransfer	External	✓	onlyOwner
	burnFeePercent	External		-
	liquidityFeePercent	External		-
	distributionFeePercent	External		-
	treasuryFeePercent	External		-
	devFeePercent	External		-
	smashTimeLotteryPrizeFeePercent	Public		-
	holdersLotteryPrizeFeePercent	Public		-
	donationLotteryPrizeFeePercent	Public		-
	isExcludedFromFee	External		-
	isExcludedFromReward	External		-

	smashTimeLotteryEnabled	External		-
	holdersLotteryEnabled	External		-
	holdersLotteryTxTrigger	External		-
	holdersLotteryMinPercent	External		-
	donationAddress	External		-
	donationsLotteryEnabled	External		-
	minimumDonationEntries	External		-
	minimalDonation	External		-
	subscriptionId	External		-
	callbackGasLimit	External		-
	requestConfirmations	External		-
	gasPriceKey	External		-
IConfiguration	Interface			
	setConsumerConfig	External	✓	-
	setSubscriptionId	External	✓	-
	setCallbackGasLimit	External	✓	-
	setRequestConfirmations	External	✓	-
	setGasPriceKey	External	✓	-
	setTeamAddress	External	✓	-
	setTeamAccumulationAddress	External	✓	-
	setTreasuryAddress	External	✓	-
	setTreasuryAccumulationAddress	External	✓	-

	setFeeConfig	External	✓	-
	switchSmashTimeLotteryFlag	External	✓	-
	switchHoldersLotteryFlag	External	✓	-
	switchDonationsLotteryFlag	External	✓	-
	excludeFromFee	External	✓	-
	includeInFee	External	✓	-
	setHoldersLotteryTxTrigger	External	✓	-
	setHoldersLotteryMinPercent	External	✓	-
	setDonationAddress	External	✓	-
	setMinimalDonation	External	✓	-
	setFee	External	✓	-
	setMinimumDonationEntries	External	✓	-
	burnFeePercent	External		-
	liquidityFeePercent	External		-
	distributionFeePercent	External		-
	treasuryFeePercent	External		-
	devFeePercent	External		-
	smashTimeLotteryPrizeFeePercent	External		-
	holdersLotteryPrizeFeePercent	External		-
	donationLotteryPrizeFeePercent	External		-
	isExcludedFromFee	External		-
	isExcludedFromReward	External		-
	smashTimeLotteryEnabled	External		-

	holdersLotteryEnabled	External		-
	holdersLotteryTxTrigger	External		-
	holdersLotteryMinPercent	External		-
	donationAddress	External		-
	donationsLotteryEnabled	External		-
	minimumDonationEntries	External		-
	minimalDonation	External		-
	subscriptionId	External		-
	callbackGasLimit	External		-
	requestConfirmations	External		-
	gasPriceKey	External		-

Inheritance Graph



Flow Graph

See the detailed images in the github repository.

Summary

Cultiv8 is an interesting project that has a friendly and growing community. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope'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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io