Cyberscope

Audit Report

Lottery

March 2023

Address	423f9683f79f53bb88a26c08367a0d1228dcfc594291579db2877200f172284b 6fef72d2380e4d980405f485d02a1ea80a1c666e8ae2c2b8948467ce79658df9
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Testing Deploy
AeternaLottery	https://testnet.bscscan.com/address/0xd2bbd7c9f28123b0729f1a6da796ee9d4450dfee
RandomNumberGenerator	https://testnet.bscscan.com/address/0x1710B448779653A13c280E8426C18260bF8e2270#code

Audit Updates

Initial Audit	13 Feb 2023
Corrected Phase 2	21 Feb 2023
Corrected Phase 3	15 Mar 2023

Source Files

Filename	SHA256
lottery.sol	423f9683f79f53bb88a26c08367a0d1228dcfc594291579db2877200f172284b
RNG.sol	6fef72d2380e4d980405f485d02a1ea80a1c666e8ae2c2b8948467ce79658df9

Introduction

This audit is focused on the Lottery contract and the RGN contract.

Lottery

The Lottery contract implements a lottery mechanism.

Lottery Mechanism Description

Only one lottery event can occur at a time, and once it is finished, the next event can start. Each lottery ticket contains six unique numbers ranging from 0 to 65. Users have the freedom to purchase as many tickets as they desire, and if they possess Aetera tokens, they will receive a 50% discount on each ticket. The winners will receive 60% of the total collected amount from the bought tickets. The results are drawn utilizing Chainlink oracle to ensure true random numbers.

Lottery State

The lottery states consists of four states:

- Pending
- Open
- Close
- Claimable

Roles

The contract roles consists of three roles. The owner, owerOrInjected, and operator roles.

The `Owner` has the authority to:

- Set contract multiplier
- Change random generator address.
- Recover lost tokens.
- Configure contract addresses like Operator, Treasury, Injector, and Wallet addresses.

The `OwnerOrInjected` has the authority to:

- Inject funds to a lottery.

The `Operator` has the authority to:

- Start a lottery.
- Close a lottery.
- Draw the final number for a lottery and make the lottery claimable.

The `Users` have the authority to:

- Buy tickets.
- Claim rewards from tickets.
- View current LotteryId.
- View a specific Lottery.
- View numbers and statuses for TicketIds.
- view rewards for TicketId.
- View user information for LotteryId.

RGN

The RandomNumberGenerator contract integrates the Chainlinks VRF Contract into the ecosystem.

Oracle Chain Review

In order for a chain oracle to function properly, it must have sufficient funds to cover the cost of making transactions on the blockchain. Without these funds, the oracle may not be able to perform its intended functions or could become stuck in a state of inactivity. Therefore, it is crucial to ensure that the necessary funds are available for the chain oracle to operate smoothly.

Roles

The contract roles consist of the owner and the LotteryAddress role.

The `Owner` has the authority to:

- Set VRF fee.
- Set VRF key hash.
- Set Lottery Key address.
- Withdraw tokens.

The `LotteryAddress` has the authority to:

- Get a random number.

The `Users` have the authority to:

- View the latest lottery id.
- View the generated random number.

Diagnostics

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	DSM	Data Structure Misuse	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	RNCM	Random Number Contract Mocking	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	lottery.sol#L323,767,1268
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 contract utilized two identical methods. Hence, one of them is redundant.

```
function isContract(address account) internal view returns (bool)
function _isContract(address _addr) internal view returns (bool)
```

The contract performs a redundant calculation. Because the variable `_lotteries[_lotteryId].rewardPerBracket[i] = 0` is set to zero the following calculations will always aggregate to zero `(_lotteries[_lotteryId].rewardsBreakdown[i] * amountToShareToWinners) /10000;`.

```
_lotteries[_lotteryId].rewardPerBracket[i] = 0;
    amountToWithdrawToTreasury +=
        (_lotteries[_lotteryId].rewardsBreakdown[i] *
            amountToShareToWinners) /
        10000;
```

Recommendation

The team is advised to take into consideration these segments 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.

- The contract could remove redundant functions.
- The contract could remove redundant calculations.

DSM - Data Structure Misuse

Criticality	Minor / Informative
Location	lottery.sol#L1365
Status	Unresolved

Description

The contract uses the valuable `_rewardsBreakdown` as an array. The business logic of the contract does not utilize the first three elements. Thus, the first three elements are redundant.

```
_rewardsBreakdown[0] = 0;
_rewardsBreakdown[1] = 0;
_rewardsBreakdown[2] = 0;
_rewardsBreakdown[3] = 100;
_rewardsBreakdown[4] = 1000;
_rewardsBreakdown[5] = 8900;
```

Recommendation

The contract could modify the way that accesses the data structure in order to remove redundant elements from the array.

For instance, the contract could utilize an offset. And as a result, less space will be utilized which leads to less gas consumption.

```
if (matchCount >= 4) {
    return _lotteries[_lotteryId].rewardPerBracket[matchCount - 4];
} else {
    return 0;
}
```

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	lottery.sol#L710
Status	Unresolved

Description

The contract is using the variables `currentLotteryId` and `currentTicketId` to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint256 public currentLotteryId;
uint256 public currentTicketId;
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

The variable `amountToWallets` may not be splitted as expected.

```
payable(wallet1).transfer(amountToWallets * 100 / 1500);
payable(wallet2).transfer(amountToWallets * 100 / 1500);
payable(wallet3).transfer(amountToWallets * 100 / 1500);
payable(wallet4).transfer(amountToWallets * 500 / 1500);
payable(wallet5).transfer(amountToWallets * 500 / 1500);
payable(wallet6).transfer(amountToWallets * 25 / 1500);
payable(wallet7).transfer(amountToWallets * 25 / 1500);
payable(wallet8).transfer(amountToWallets * 25 / 1500);
payable(wallet9).transfer(amountToWallets * 25 / 1500);
payable(wallet10).transfer(amountToWallets * 25 / 1500);
payable(wallet11).transfer(amountToWallets * 75 / 1500);
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

RNCM - Random Number Contract Mocking

Criticality	Minor / Informative
Location	lottery.sol#L1090
Status	Unresolved

Description

The contract is prone to contract mocking, as the `randomGenerator` contract it relies on can be changed. The `_randomGeneratorAddress` argument used by the contract is unverified, and this can potentially lead to security issues that could negatively impact transactions. For example, it may allow for the manipulation of random numbers, compromising the integrity of the contract's operations.

```
function changeRandomGenerator(
    address _randomGeneratorAddress
) external onlyOwner {
    require(
        _lotteries[currentLotteryId].status == Status.Claimable,
        "Lottery not in claimable"
    );
    // Request a random number from the generator based on a seed
    IRandomNumberGenerator(_randomGeneratorAddress).getRandomNumber(
        uint256(
            keccak256(abi.encodePacked(currentLotteryId, currentTicketId))
        )
    );
    // Calculate the finalNumber based on the randomResult generated by ChainLink's
fallback
    IRandomNumberGenerator(_randomGeneratorAddress).viewRandomResult();
    randomGenerator = IRandomNumberGenerator(_randomGeneratorAddress);
    emit NewRandomGenerator(_randomGeneratorAddress);
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	lottery.sol#L917,920
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
aeternaToke
priceFee
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	RNG.sol#L650,651,652,653,671,672,822,823,824,846,1016,1028,1036,1044,1059,1060 lottery.sol#L799,837,942,956,967,979,980,1066,1095,1096,1155,1178,1204,1205,1319,1347,1364,1441,1442,1462,1463,1464,1465,1508,1518,1543,1544,1570,1571,1572,1573
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.

```
bytes32 _keyHash
uint256 _userSeed
address _requester
uint256 _nonce
uint256 _vRFInputSeed
uint256 _fee
uint256 _seed
LinkTokenInterface internal immutable LINK
address _pancakeSwapLottery
address _tokenAddress
uint256 _tokenAmount
uint256 public player_count
TicketNumber private INIT_TICKET_VALUE
TicketNumber memory _ticket

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	RNG.sol#L252,284,316,361,379,397,415,474,493,512,528 lottery.sol#L352,384,416,461,479,497,515,574,593,612,628
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount) internal {
        require(
            address(this).balance >= amount,
            "Address: insufficient balance"
        );

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{value: amount}("");
        require(
            success,
            "Address: unable to send value, recipient may have reverted"
        );
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	lottery.sol#L1236,1261,1269
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.

```
uint256 amountToShareToWinners = (
            ((_lotteries[_lotteryId].amountCollected) * 6000)
        ) / 10000
amountToWithdrawToTreasury +=
                    (_lotteries[_lotteryId].rewardsBreakdown[i] *
                        amountToShareToWinners) /
                    10000
```

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.

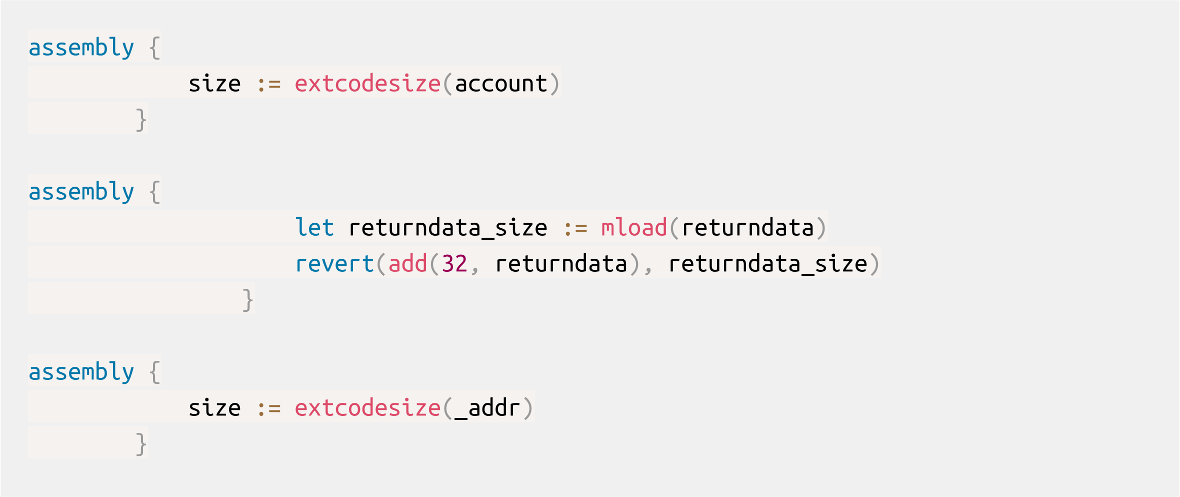
L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	RNG.sol#L230,440 lottery.sol#L330,540,1646
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 {
            size := extcodesize(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }

assembly {
            size := extcodesize(_addr)
        }
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	RNG.sol#L9,34,108,200,453,579,632,680,882,903,981 lottery.sol#L9,34,108,169,208,300,553,679,700,769,770
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.4;

pragma solidity ^0.8.0;
pragma solidity ^0.8.4;
pragma abicoder v2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	RNG.sol#L9,34,108,200,453,579,632,680,882,903,981 lottery.sol#L9,34,108,169,208,300,553,679,700,769
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
ReentrancyGuard	Implementation			
		Public	✓	-
AggregatorV3Interface	Interface			
	decimals	External		-
	description	External		-
	version	External		-
	getRoundData	External		-
	latestRoundData	External		-
IERC20	Interface			
	totalSupply	External		-

	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
IRandomNumberGenerator	Interface			
	getRandomNumber	External	✓	-

	viewLatestLotteryId	External		-
	viewRandomResult	External		-
IAeternaLottery	Interface			
	buyTickets	External	Payable	-
	claimTickets	External	✓	-
	closeLottery	External	✓	-
	drawFinalNumberAndMakeLotteryClaimable	External	✓	-
	injectFunds	External	Payable	-
	startLottery	External	✓	-
	viewCurrentLotteryId	External	✓	-
AeternaLottery	Implementation	ReentrancyGuard, IAeternaLottery, Ownable		
		Public	✓	-
	getLatestPrice	Public		-
	checkTicket	Internal		
	matchTicket	Internal		
	setMultiplier	External	✓	onlyOwner
	buyTickets	External	Payable	notContract nonReentrant
	airdropTickets	External	✓	onlyOwner nonReentrant
	redeemTicket	Public	✓	nonReentrant
	claimTickets	External	✓	notContract nonReentrant
	closeLottery	External	✓	onlyOperator nonReentrant
	finalizeWinningNumber	Internal	✓	
	drawFinalNumberAndMakeLotteryClaimable	External	✓	onlyOperator nonReentrant

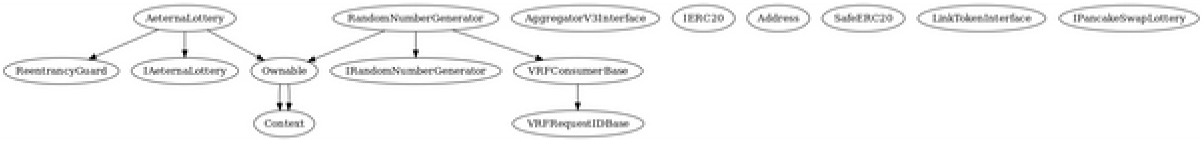
	changeRandomGenerator	External	✓	onlyOwner
	injectFunds	External	Payable	onlyOwnerOrInjector
	startLottery	External	✓	onlyOperator
	recoverWrongTokens	External	✓	onlyOwner
	setOperatorAndTreasuryAndInjectorAddresses	External	✓	onlyOwner
	viewCurrentLotteryId	External		-
	viewLottery	External		-
	viewNumbersAndStatusesForTicketIds	External		-
	viewRewardsForTicketId	External		-
	viewUserInfoForLotteryId	External		-
	_calculateRewardsForTicketId	Internal		
	_isContract	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-

	approve	External	✓	-
	transferFrom	External	✓	-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
LinkTokenInterface	Interface			
	allowance	External		-
	approve	External	✓	-
	balanceOf	External		-
	decimals	External		-

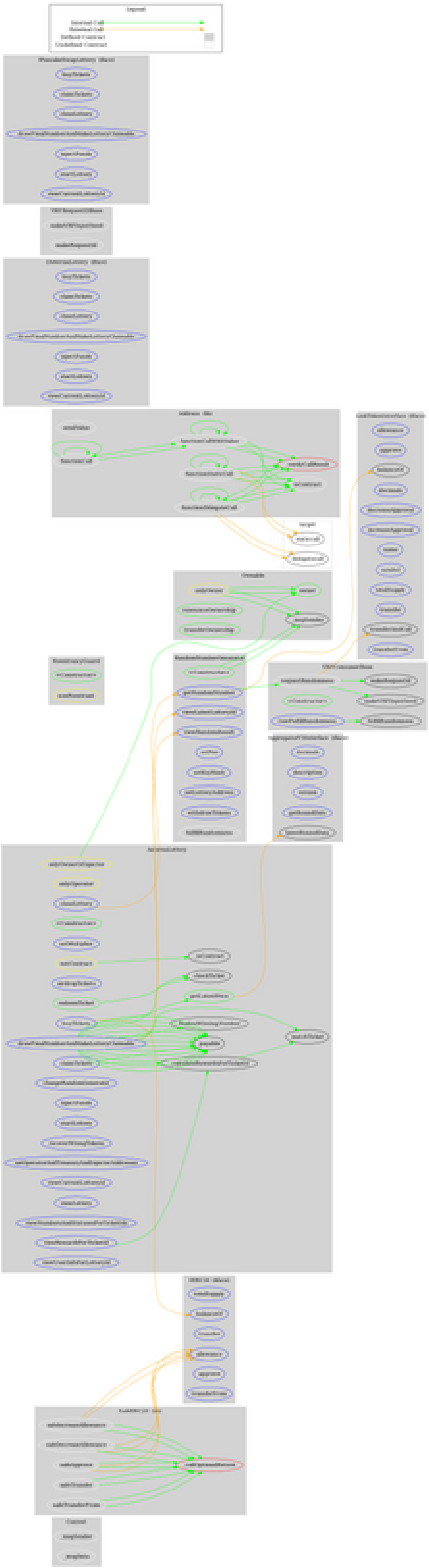
	decreaseApproval	External	✓	-
	increaseApproval	External	✓	-
	name	External		-
	symbol	External		-
	totalSupply	External		-
	transfer	External	✓	-
	transferAndCall	External	✓	-
	transferFrom	External	✓	-
VRFRequestIDBase	Implementation			
	makeVRFInputSeed	Internal		
	makeRequestId	Internal		
VRFConsumerBase	Implementation	VRFRequestIDBase		
	fulfillRandomness	Internal	✓	
	requestRandomness	Internal	✓	
		Public	✓	-
	rawFulfillRandomness	External	✓	-
IRandomNumberGenerator	Interface			
	getRandomNumber	External	✓	-
	viewLatestLotteryId	External		-
	viewRandomResult	External		-
IPancakeSwapLottery	Interface			
	buyTickets	External	✓	-
	claimTickets	External	✓	-
	closeLottery	External	✓	-

	drawFinalNumberAndMakeLotteryClaimable	External	✓	-
	injectFunds	External	✓	-
	startLottery	External	✓	-
	viewCurrentLotteryId	External	✓	-
RandomNumberGenerator	Implementation	VRFConsumerBase, IRandomNumberGenerator, Ownable		
		Public	✓	VRFConsumerBase
	getRandomNumber	External	✓	-
	setFee	External	✓	onlyOwner
	setKeyHash	External	✓	onlyOwner
	setLotteryAddress	External	✓	onlyOwner
	withdrawTokens	External	✓	onlyOwner
	viewLatestLotteryId	External		-
	viewRandomResult	External		-
	fulfillRandomness	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract

Summary

Lottery contract implements a lottery and financial mechanism. 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