Cyberscope

Audit Report

Addictive Crypto Games

May 2023

SHA256 a50517d50b7913ba0f7cc6ae5b8ee8013a5fe6bdb3924c78c82ccc48fe936a3d

Audited by © cyberscope

Table of Contents

Review

Contract Name	ACG
Testing Deploy	https://explorer-mumbai.maticvigil.com/address/0x38b8d3e7c6a27e1de1931a9a31a40b0a14b30182
Symbol	ACG
Decimals	18
Total Supply	1.000.000.000

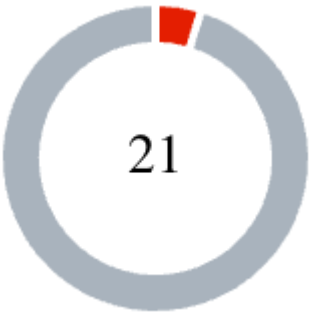
Audit Updates

Initial Audit	23 May 2023

Source Files

Filename	SHA256
contracts/acg.sol	a50517d50b7913ba0f7cc6ae5b8ee8013a5fe6bdb3924c78c82ccc48fe936a3d
contracts/contracts/access/Ownable.sol	42df7a70b8190e7c8e3aeb443aeacc2b23b389b18fa2ce00e9eb60a367a2bb20
contracts/contracts/interfaces/IUniswapV2Factory.sol	3dd4c1f051cee242d1c81b3868d19d983706f47dc6d4e61c83e8645dab7b190f
contracts/contracts/interfaces/IUniswapV2Pair.sol	d031a0cf0541e16cc08a0772453796dcbf77727976822ac038dbea47e16171cb
contracts/contracts/interfaces/IUniswapV2Router01.sol	6074134cb0a1a8be61b74be39d8e1539e8d6cd28d39d25aee0b6a2d9beba2d3c
contracts/contracts/interfaces/IUniswapV2Router02.sol	add2f9ec336a24dfe0fcf25cd27fd11860fa09f8e303867f5188b2b1769b31e4
contracts/contracts/token/ERC20/ERC20.sol	bce14c3fd3b1a668529e375f6b70ffdf9cef8c4e410ae99608be5964d98fa701
contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
contracts/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
contracts/contracts/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e81f0436689132a4e73394bab084a6d
contracts/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
contracts/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca0dfe5ec14916369a2935d888ff257a

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	20	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
	ST	Stops Transactions	Unresolved
	OCTD	Transfers Contract's Tokens	Passed
	OTUT	Transfers User's Tokens	Passed
	ELFM	Exceeds Fees Limit	Unresolved
	ULTW	Transfers Liquidity to Team Wallet	Passed
	MT	Mints Tokens	Passed
	BT	Burns Tokens	Passed
	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Critical
Location	contracts/acg.sol#L1208
Status	Unresolved

Description

The contract owner has the authority to stop buy transactions for all users. The owner may take advantage of it by setting the `_maxBotSellCount` to zero and `_botSellTimeBlockLimit` to a high value. As a result, the contract may operate as a honeypot.

```
if (to == uniswapV2Pair && antiBotEnabled) {
  bool canTransfer = _checkCanTransfer(from);
  if (!canTransfer) {
      return; // Exit, Bot detected.
  }
}
```

Recommendation

The contract could embody a check for not allowing setting the _maxBotSellCount less than a reasonable amount and _botSellTimeBlockLimit more than a reasonable value. A suggested implementation could check that the maximum 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	contracts/acg.sol#L811
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setPenaltyFeePercent` function with a high percentage value.

```
function setPenaltyFeePercent(
  uint256 newfee
) external onlyOwner feesNotBeingSet {
  ...
  // Ensure the total penalty fees do not exceed 20%
  require(newfee <= 20, "ACG: Total penalty fees should not exceed 20%.");
  ...
}

function _calculateSellFee(uint256 tAmount) private view returns (uint256) {
  ...

  if (tAmount >= _penaltyTxAmount) {
      _sellFees = _sellFees.add(_tPenaltyFee);
  }

  return _sellFees;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RSC	Redundant Swap Condition	Unresolved
●	RFC	Redundant Fee Calculation	Unresolved
●	PAP	Pair Address Prevalidation	Unresolved
●	FO	Function Optimization	Unresolved
●	RV	Redundant Variable	Unresolved
●	TUU	Time Units Usage	Unresolved
●	MSE	Missing Solidity Events	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	CR	Code Repetition	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

RSC - Redundant Swap Condition

Criticality	Minor / Informative
Location	contracts/acg.sol#L1219
Status	Unresolved

Description

As part of the swap precondition logic, the contract normalizes the `contractTokenBalance` if it proceeds the `_swapThresholdLimit`. In the next statement, it proceeds with the same condition. Since the condition already exists in the first statement, then the second is redundant.

```
// Set contract's token balance to maxTxAmount if it is greater
than maxTxAmount
if (contractTokenBalance >= _swapThresholdLimit) {
    contractTokenBalance = _swapThresholdLimit;
}

// Check if the contract's token balance is over the swap threshold
limit
bool overSwapThresholdLimit = contractTokenBalance >=
    _swapThresholdLimit;
```

Recommendation

The team is adviced to merge the `overSwapThresholdLimit` with the `contractTokenBalance` normalization logic to avoid the redundant condition in every transfer.

RFC - Redundant Fee Calculation

Criticality	Minor / Informative
Location	contracts/acg.sol#L966
Status	Unresolved

Description

The contract calculates the `_tPenaltyFee` on every transfer but it is only used when the expression `tAmount >= _penaltyTxAmount` is fullyfied. This statement increases unnecessarily the gas consuption.

```
uint256 _tPenaltyFee = _calculateFee(tAmount, _penaltyFee);

uint256 _sellFees = _tPoolsLeaderboardFee
    .add(_tCommunityFee)
    .add(_tTreasuryOneFee)
    .add(_tTreasuryTwoFee);

if (tAmount >= _penaltyTxAmount) {
    _sellFees = _sellFees.add(_tPenaltyFee);
}
```

Recommendation

The team is advised to calculate the penalty fees only when it is required.

PAP - Pair Address Prevalidation

Criticality	Minor / Informative
Location	contracts/acg.sol#L235,1520
Status	Unresolved

Description

The variable `erc20TokenFeeAddress` can be any address. Additionally, the contract does not validate if a token pair exists between the following four addresses. This lack of validation can lead to unintended behavior and potential security vulnerabilities.

```
erc20TokenFeeAddress = address(
    0xEEd229C1d444fA45E750D8527C0dc27323020ddB
);

function _swapTokensForTokens(uint256 tokenAmount) private {
    address[] memory path = new address[](3);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    path[2] = address(erc20TokenFeeAddress);
    ...
}
```

Recommendation

It is recommended to perform a prevalidation check on the contract addresses used for swapping, to ensure a smooth transaction flow within the contract. This validation should confirm that the addresses have valid pair address values associated with them.

FO - Function Optimization

Criticality	Minor / Informative
Location	contracts/acg.sol#L1167
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 could decrease the operation of the `_checkCanTransfer` method. Since the if condition is redundant.

```
function _checkCanTransfer(address from) private view returns
(bool) {
    uint256 _reqLimit = botLastTransferBlock[from].add(
        _botSellTimeBlockLimit
    );
    // uint256 _currentBlockNumber = block.number;

    if (
        botSellCount[from] >= _maxBotSellCount && block.number <
_reqLimit
    ) {
        return false;
    }

    return true;
}
```

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.

The contract could return directly the conditional statement. For instance,

```
function _checkCanTransfer(address from) private view returns
(bool) {
    ...
    return (!(botSellCount[from] >= _maxBotSellCount &&
block.number < _reqLimit))
}
```

RV - Redundant Variable

Criticality	Minor / Informative
Location	contracts/acg.sol#L1071
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 variable `tFee` does not serve any purpose in the function implementation. Hence, the variable is redundant.

```
function _getTValues(
    uint256 tAmount,
    address recipient
) private view returns (uint256, uint256) {
    uint256 _totalReqFees = _calculateFees(tAmount, recipient);

    uint256 tFee = _totalReqFees;

    uint256 tTransferAmount = tAmount.sub(tFee);

    return (tTransferAmount, tFee);
}
```

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. It is recommended to remove redundant variables.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/acg.sol#L501
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 _newSellTimeLimitInBlocks = _newSellTimeLimitInMinutes
    .mul(60)
    .div(3); // 1 block every 3 seconds
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days`, `weeks` and `years` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	contracts/acg.sol#L,467,480,514,518,522,526,530,534,538,546,584,621,659,597,735,773,811,899,909
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 setErc20TokenAddress(address newToken) external onlyOwner
{
    require(newToken != address(0), "ACG: Invalid ERC-20 token
address");
    require(
        erc20TokenFeeAddress != newToken,
        "ACG: Token address is the same."
    );
    erc20TokenFeeAddress = address(newToken);
}

function setAntiBotEnabled(bool _option) external onlyOwner {
    require(_option != antiBotEnabled, "Can't set the same
option");
    antiBotEnabled = _option;
}

...
```

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/acg.sol#L74,459
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.

Since the variable `_penaltyTxAmount` is meant for anti whale mechanism. The value of the variable should not be able to be set to zero.

```
// The whale amount of tokens that is being swapped in a single
transaction.
uint256 public _penaltyTxAmount = 10000000 * 10 ** 18; // 1% of the
total supply

function setPenaltyTxAmount(uint256 _newLimit) external onlyOwner {
    _penaltyTxAmount = _newLimit;
}
```

Recommendation

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

- The variable `_penaltyTxAmount` should be greater than a percentage of the total supply.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/acg.sol#L451
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `_swapThresholdLimit` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setSwapThresholdLimit(uint256 _newLimit) external
onlyOwner {
    _swapThresholdLimit = _newLimit;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/acg.sol#L556,597,635,673,711,749,787
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
require(totalNewFees <= 15, "ACG: Total fees should not exceed
15%.");

...

// Update the total required buy fees to reflect the new fee
configuration
totalRequiredBuyFees = totalNewFees;
// Update the total required fees to reflect the new fee
configuration
totalRequiredFees = totalNewFees.add(totalRequiredSellFees).add(
    _penaltyFee
);
// Update the total ERC20 fees to reflect the new fee configuration
// Subtract the liquidity fee as it is handled separately
totalERC20Fees = totalNewFees
    .add(totalRequiredSellFees)
    .add(_penaltyFee)
    .sub(_liquidityFee);
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/acg.sol#L430,451,459,514,518,522,526,530,534,538,899,909,919
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 updates the router address even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function setSwapRouter(address newRouter) external onlyOwner {
    require(newRouter != address(0), "ACG: Invalid router
address");
    IUniswapV2Factory factory = IUniswapV2Factory(newRouter);
    address wethAddress = IUniswapV2Router02(newRouter).WETH();

    // Check if a pair already exists for this token and WETH on
the new router
    address pairAddress = factory.getPair(address(this),
wethAddress);
    if (pairAddress == address(0)) {
        // If no pair exists, create a new one
        pairAddress = factory.createPair(address(this),
wethAddress);
    }

    // Update the router and pair addresses
    uniswapV2Router = IUniswapV2Router02(newRouter);
    uniswapV2Pair = pairAddress;
}
```

The contract updates the addresses and values even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function setSwapThresholdLimit(uint256 _newLimit) external
onlyOwner {
    _swapThresholdLimit = _newLimit;
}

function setPenaltyTxAmount(uint256 _newLimit) external onlyOwner {
    _penaltyTxAmount = _newLimit;
}

function setOperationWallet(address _newWallet) external onlyOwner
{
    operationWallet = _newWallet;
}

function setMarketingWallet(address _newWallet) external onlyOwner
{
    marketingWallet = _newWallet;
}

function setPoolsLeaderboardWallet(address _newWallet) external
onlyOwner {
    poolsLeaderboardWallet = _newWallet;
}

function setCommunityWallet(address _newWallet) external onlyOwner
{
    communityWallet = _newWallet;
}

function setTreasuryOneWallet(address _newWallet) external
onlyOwner {
    treasuryOneWallet = _newWallet;
}

function setTreasuryTwoWallet(address _newWallet) external
onlyOwner {
    treasuryTwoWallet = _newWallet;
}

function setPenaltyWallet(address _newWallet) external onlyOwner {
    penaltyWallet = _newWallet;
}

function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

```
function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
    _isSwapAndGetFeesEnabled = _enabled;
    emit SwapAndLiquifyEnabledUpdated(_enabled);
}
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/acg.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/acg.sol#L64,72,84,86,88
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 _totalSupply = 1000000000 * 10 ** 18
uint256 public _maxTxAmount = 150000000 * 10 ** 18
string private _name = "Addictive Crypto Games"
string private _symbol = "ACG"
uint8 private _decimals = 18
```

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/acg.sol#L58,59,72,74,77,98,104,108,112,116,120,124,128,132,421,451,459,480,489,499,514,518,522,526,530,534,538,919,1609,1629,1630,1652,1653,1676,1677,1678
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.

```
uint256 public _maxBotSellCount = 5
uint256 public _botSellTimeBlockLimit = 100
uint256 public _maxTxAmount = 150000000 * 10 ** 18
uint256 public _penaltyTxAmount = 10000000 * 10 ** 18
uint256 public _totalFeesCharged
bool public _isSwapAndGetFeesEnabled = true
uint256 public _operationFee = 2
uint256 public _marketingFee = 3
uint256 public _poolsLeaderboardFee = 5
uint256 public _communityFee = 1
uint256 public _treasuryOneFee = 2
uint256 public _treasuryTwoFee = 2
uint256 public _liquidityFee = 5
uint256 public _penaltyFee = 10

...
```

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/acg.sol#L452,460,491,508,564,601,639,677,715,753,791,825
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.

```
_swapThresholdLimit = _newLimit
_penaltyTxAmount = _newLimit
_maxBotSellCount = _newCount
_botSellTimeBlockLimit = _newSellTimeLimitInBlocks
_operationFee = newfee
_marketingFee = newfee
_poolsLeaderboardFee = newfee
_communityFee = newfee
_treasuryOneFee = newfee
_treasuryTwoFee = newfee
_liquidityFee = newfee
_penaltyFee = newfee
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/acg.sol#L1125
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 _takeLiquidity(uint256 tLiquidity) private {
        uint256 currentRate = _getRate();
        uint256 rLiquidity = tLiquidity.mul(currentRate);
        _balances[address(this)] =
_balances[address(this)].add(rLiquidity);
    }
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/acg.sol#L515,519,523,527,531,535,539
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
operationWallet = _newWallet
marketingWallet = _newWallet
poolsLeaderboardWallet = _newWallet
communityWallet = _newWallet
treasuryOneWallet = _newWallet
treasuryTwoWallet = _newWallet
penaltyWallet = _newWallet
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/acg.sol#L2
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;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/acg.sol#L1461,1660,1686
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(erc20TokenFeeAddress).transfer(_wallet, _amount)
IERC20(_token).transfer(owner(), _amount)
IERC20(_token).transfer(_to, _amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

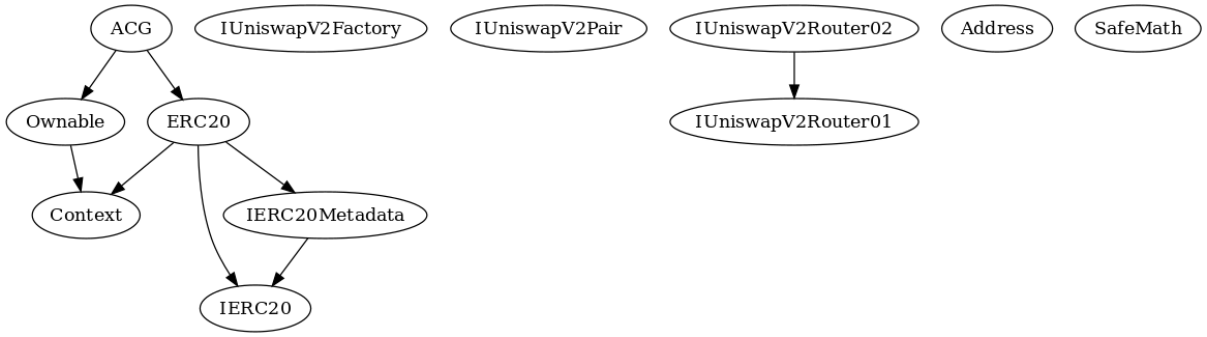
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ACG	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
		External	Payable	-
	_approve	Internal	✓	
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	getBNBBalance	Public		-
	getErc20TokenFeeBalance	Public		-
	getErc20TokensBalance	Public		-
	setSwapRouter	External	✓	onlyOwner

	setSwapThresholdLimit	External	✓	onlyOwner
	setPenaltyTxAmount	External	✓	onlyOwner
	setErc20TokenAddress	External	✓	onlyOwner
	setAntiBotEnabled	External	✓	onlyOwner
	setMaxBotSellCount	External	✓	onlyOwner
	setBotSellTimeLimit	External	✓	onlyOwner
	setOperationWallet	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setPoolsLeaderboardWallet	External	✓	onlyOwner
	setCommunityWallet	External	✓	onlyOwner
	setTreasuryOneWallet	External	✓	onlyOwner
	setTreasuryTwoWallet	External	✓	onlyOwner
	setPenaltyWallet	External	✓	onlyOwner
	setOperationFeePercent	External	✓	onlyOwner feesNotBeingSet
	setMarketingFeePercent	External	✓	onlyOwner feesNotBeingSet
	setPoolsLeaderboardFeePercent	External	✓	onlyOwner feesNotBeingSet
	setCommunityFeePercent	External	✓	onlyOwner feesNotBeingSet
	setTreasuryOneFeePercent	External	✓	onlyOwner feesNotBeingSet
	setTreasuryTwoFeePercent	External	✓	onlyOwner feesNotBeingSet

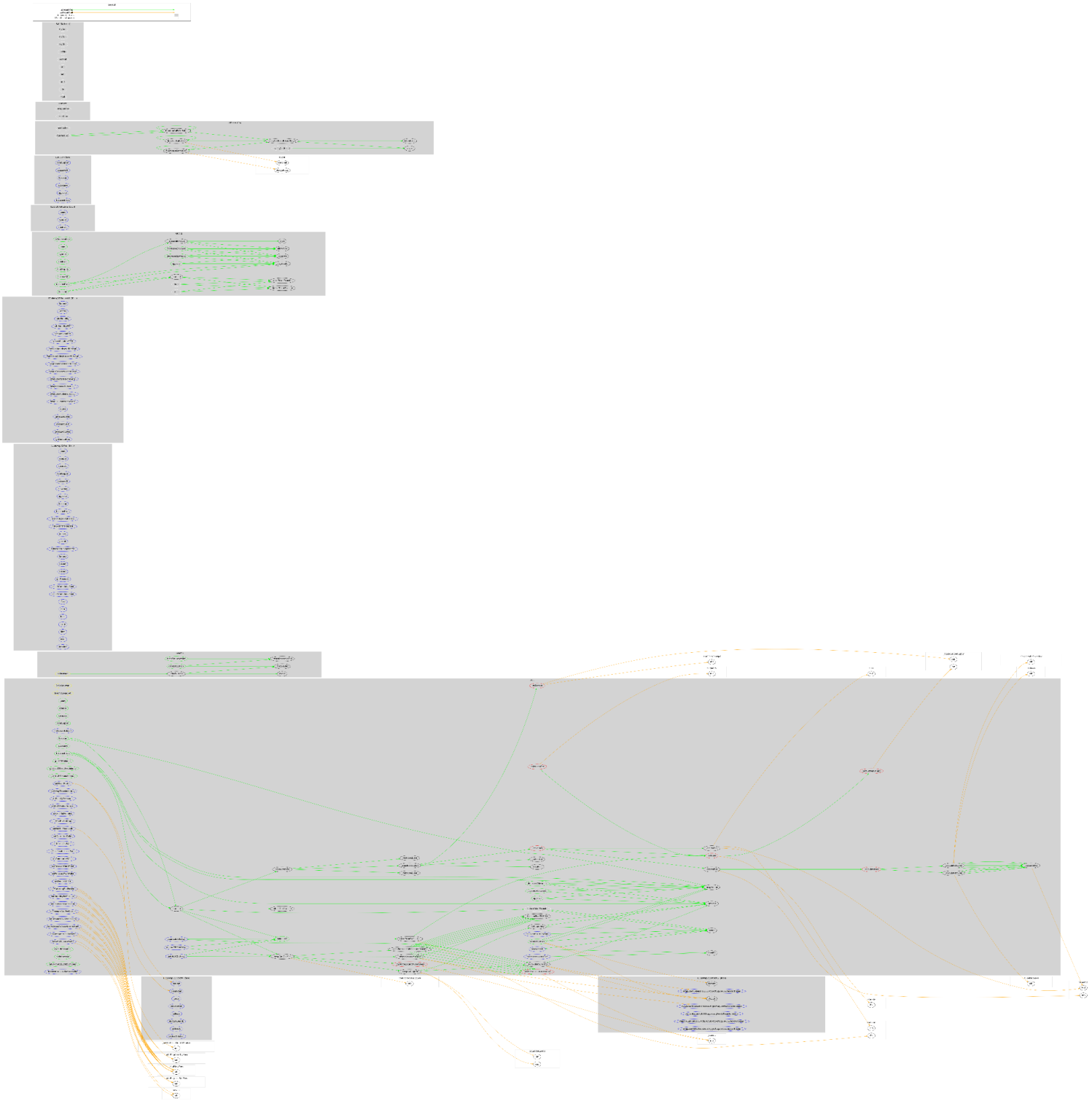
	setLiquidityFeePercent	External	✓	onlyOwner feesNotBeingSet
	setPenaltyFeePercent	External	✓	onlyOwner feesNotBeingSet
	_removeAllFees	Private	✓	
	_restoreAllFees	Private	✓	
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	_calculateFees	Private		
	_calculateBuyFee	Private		
	_calculateSellFee	Private		
	_calculateFee	Private		
	_getCurrentSupply	Private		
	_getRate	Private		
	tokenFromReflection	Public		-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_calculateERC20TokenFees	Private		
	_calculateLiquidityTokenFees	Private		
	_takeLiquidity	Private	✓	
	_takeFees	Private	✓	

	_beforeTokenTransfer	Internal		
	_checkCanTransfer	Private		
	_transfer	Internal	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_reflectBot	Private	✓	
	_swapAndGetFees	Private	✓	lockTheSwap
	_swapAndLiquify	Private	✓	
	_calculateAvailableFeesAndTransfer	Private	✓	
	_transferFeesToWallet	Private	✓	
	_swapTokensForBnb	Private	✓	
	_addLiquidity	Private	✓	
	_swapTokensForTokens	Private	✓	
	manualBNBSwap	External	✓	onlyOwner
	manualERC20Swap	External	✓	onlyOwner lockTheSwap
	autoERC20Swap	External	✓	onlyOwner
	recoverBNB	External	✓	onlyOwner
	recoverBNBToWallet	External	✓	onlyOwner
	recoverERC20Tokens	External	✓	onlyOwner
	recoverERC20TokensToWallet	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Addictive Crypto Games Token contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions and manipulating the fees. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. Additionally, the contract has an antibot throttling mechanism.

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