Cyberscope

Audit Report

GameCraft

June 2023

Network BSC

Address 0xd74bDD4547b895d798aeCC71Bd4290CA6da50F26

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	FRV	Fee Restoration Vulnerability	Unresolved
Minor / Informative	CO	Calculation Optimization	Unresolved
Minor / Informative	RE	Redundant Events	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	FSA	Fixed Swap Address	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	GCT
Compiler Version	v0.6.12+commit.27d51765
Optimization	200 runs
Explorer	https://bscscan.com/address/0xd74bdd4547b895d798aecc71bd4290ca6da50f26
Address	0xd74bdd4547b895d798aecc71bd4290ca6da50f26
Network	BSC
Symbol	GCT
Decimals	10
Total Supply	10,000,000

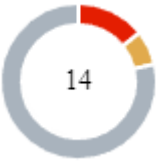
Audit Updates

Initial Audit	07 Jun 2023

Source Files

Filename	SHA256
GCT.sol	bda7d0373c0f23355d2a8b8d51a9256ad908569583a1515a510d919c0c002a88

Findings Breakdown



Critical	2
Medium	1
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	GCT.sol#L721,737
Status	Unresolved

Description

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

```
if(!isTxLimitExempt[from] && !isTxLimitExempt[to]) {
  require(amount <= _maxTxAmount, "Transfer amount exceeds the
maxTxAmount.");
}

if(checkWalletLimit && !isWalletLimitExempt[to])
  require(balanceOf(to).add(amount) <= _walletMax);
```

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting `_maxTxAmount` to a high value.

```
if(!isTxLimitExempt[from] && !isTxLimitExempt[to]) {
  require(amount <= _maxTxAmount, "Transfer amount exceeds the
maxTxAmount.");
}
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` and `_walletMax` more than a reasonable amount. 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	Critical
Location	GCT.sol#L874,878,882,886
Status	Unresolved

Description

The contract owner has the authority to increase the allowed limit of 25%. The owner may take advantage of it by calling the `setRewardFeePercent`, `setLiquidityFeePercent`, `setDeveloperFeePercent`, and `setBurnFeePercent` functions with a high percentage value.

```
function setRewardFeePercent(uint256 RewardFee) external onlyOwner() {
  _RewardFee = RewardFee;
}

function setLiquidityFeePercent(uint256 liquidityFee) external
onlyOwner() {
  _liquidityFee = liquidityFee;
}

function setDeveloperFeePercent(uint256 DeveloperFee) external
onlyOwner() {
  _DeveloperFee = DeveloperFee;
}

function setBurnFeePercent(uint256 burnFee) external onlyOwner() {
  _burnFee = burnFee;
}
```

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.

FRV - Fee Restoration Vulnerability

Criticality	Medium
Location	GCT.sol#L668,682,793
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when both _taxFee and _redisFee have been set to zero. During a transaction, if the takeFee variable is false, then both `removeAllFee` and `restoreAllFee` function will be executed. The `removeAllFee` function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent `restoreAllFee` function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the removed fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
function removeAllFee() private {
    if(_RewardFee == 0 && _liquidityFee == 0 && _DeveloperFee==0 &&
_burnFee==0) return;

    _previousRewardFee = _RewardFee;
    _previousLiquidityFee = _liquidityFee;
    _previousBurnFee = _burnFee;
    _previousDeveloperFee = _DeveloperFee;

    _RewardFee = 0;
    _liquidityFee = 0;
    _DeveloperFee = 0;
    _burnFee = 0;
}

function restoreAllFee() private {
    _RewardFee = _previousRewardFee;
    _liquidityFee = _previousLiquidityFee;
    _burnFee = _previousBurnFee;
    _DeveloperFee = _previousDeveloperFee;
}

function _tokenTransfer(address sender, address recipient, uint256
amount) private {
    if(_isExcludedFromFee[sender] || _isExcludedFromFee[recipient]){
        removeAllFee();
    }
    ...
    if(_isExcludedFromFee[sender] || _isExcludedFromFee[recipient])
        restoreAllFee();
}
```

Recommendation

The team is advised to modify the `removeAllFee` function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

```
function removeAllFee() private {
    _previousRewardFee = _RewardFee;
    _previousLiquidityFee = _liquidityFee;
    _previousBurnFee = _burnFee;
    _previousDeveloperFee = _DeveloperFee;

    _RewardFee = 0;
    _liquidityFee = 0;
    _DeveloperFee = 0;
    _burnFee = 0;
}
```

CO - Calculation Optimization

Criticality	Minor / Informative
Location	GCT.sol#L802
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 is using the same calculation on multiple lines `amount.sub(burnAmt).sub(DeveloperAmt)` . Hence, it is duplicating it.

```
if (_isExcluded[sender] && !_isExcluded[recipient]) {
    _transferFromExcluded(sender, recipient,
(amount.sub(burnAmt).sub(DeveloperAmt)));
} else if (!_isExcluded[sender] && _isExcluded[recipient]) {
    _transferToExcluded(sender, recipient,
(amount.sub(burnAmt).sub(DeveloperAmt)));
} else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
    _transferStandard(sender, recipient,
(amount.sub(burnAmt).sub(DeveloperAmt)));
} else if (_isExcluded[sender] && _isExcluded[recipient]) {
    _transferBothExcluded(sender, recipient,
(amount.sub(burnAmt).sub(DeveloperAmt)));
} else {
    _transferStandard(sender, recipient,
(amount.sub(burnAmt).sub(DeveloperAmt)));
}
```

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 make the calculation one time and pass it as an argument.

RE - Redundant Events

Criticality	Minor / Informative
Location	GCT.sol#L439,440
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 `MinTokensBeforeSwapUpdated` and `SwapAndLiquifyEnabledUpdated` events are not utilized in the contract implementation. Hence, they are redundant.

```
event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
event SwapAndLiquifyEnabledUpdated(bool 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. It is recommended to remove redundant events.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	GCT.sol#L701,705,709,862,866,870
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 variables 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 setWalletLimit(uint256 newLimit) external onlyOwner {
    _walletMax  = newLimit;
}

function enableDisableWalletLimit(bool newValue) external onlyOwner {
    checkWalletLimit = newValue;
}

function setIsWalletLimitExempt(address holder, bool exempt) external
onlyOwner {
    isWalletLimitExempt[holder] = exempt;
}

...
```

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.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	GCT.sol#L455
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
constructor () public {
    _rOwned[_msgSender()] = _rTotal;
    IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	GCT.sol#L457,461
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.

```
uniswapV2Pair
uniswapV2Router
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	GCT.sol#L406,410,411,412,434,437
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 = 10000000 * 10**10
string private _name = "GameCraft"
string private _symbol = "GCT"
uint8 private _decimals = 10
bool public swapAndLiquifyEnabled = false
uint256 private numTokensSellToAddToLiquidity = 8000 * 10**10
```

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	GCT.sol#L216,217,233,254,414,417,420,423,424,427,428,656,662,874,882
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.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint256 public _RewardFee = 0
uint256 public _liquidityFee = 1
uint256 public _burnFee = 0
uint256 public _DeveloperFee = 1
address public DeveloperWallet =
0xEE1Cdd08027b417a2a818C82C5bB066DB3daA18c
uint256 public _walletMax =     10000000 * 10**10
uint256 public _maxTxAmount =   10000000 * 10**10
uint256 _amount
uint256 RewardFee
uint256 DeveloperFee
```

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	GCT.sol#L702,875,879,883,887,891
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.

```
_walletMax  = newLimit
_RewardFee = RewardFee
_liquidityFee = liquidityFee
_DeveloperFee = DeveloperFee
_burnFee = burnFee
_maxTxAmount = maxTxAmount
```

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	GCT.sol#L91,99,107,111,115,119,124
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 isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
...
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	GCT.sol#L871
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.

```
DeveloperWallet = 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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	GCT.sol#L95,136
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 { codehash := extcodehash(account) }

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

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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		

Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-

	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-

	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-

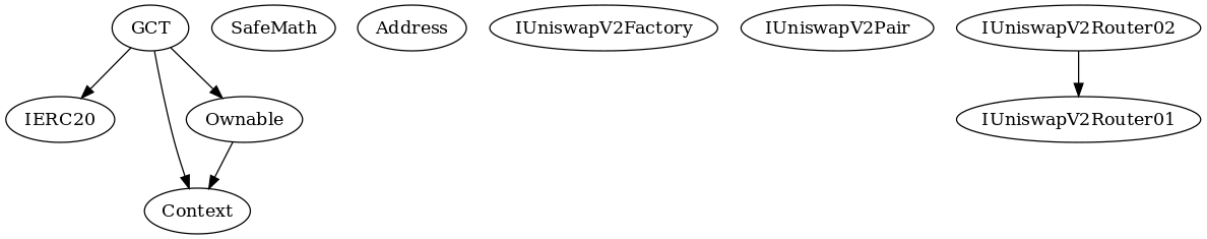
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
GCT	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-

	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	setIsTxLimitExempt	External	✓	onlyOwner
	deliver	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		

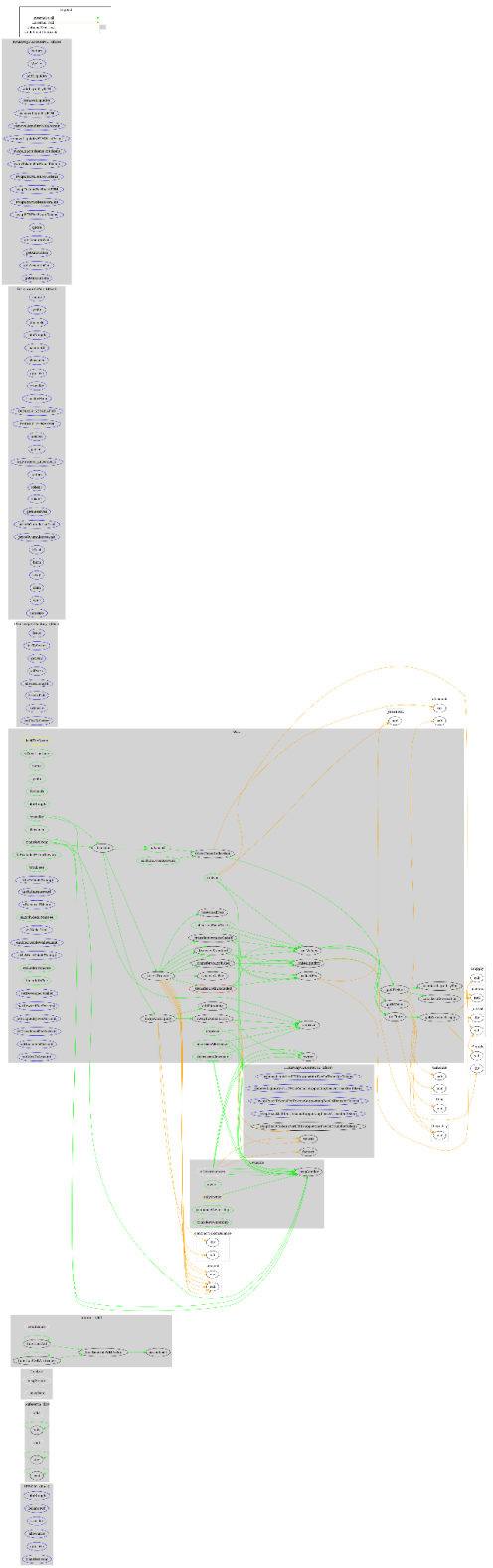
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	calculateRewardFee	Private		
	calculateLiquidityFee	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	
	setWalletLimit	External	✓	onlyOwner
	enableDisableWalletLimit	External	✓	onlyOwner
	setIsWalletLimitExempt	External	✓	onlyOwner
	_transfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setDeveloperWallet	External	✓	onlyOwner
	setRewardFeePercent	External	✓	onlyOwner

	setLiquidityFeePercent	External	✓	onlyOwner
	setDeveloperFeePercent	External	✓	onlyOwner
	setBurnFeePercent	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

GameCraft 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 stop transactions and manipulate the fees. As a result, the contract may operate as a honeypot. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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