Cyberscope

Audit Report

GameCraft

June 2023

SHA256 a216f96cb8f1b22a2d88b8ced1e0f2ae79b74bdd94d8c12082cc0d02d00afe45

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
Critical	SFV	Swap Functionality Vulnerability	Unresolved
Minor / Informative	RCS	Redundant Conditional Statement	Unresolved
Minor / Informative	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor / Informative	MEM	Misleading Error Messages	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	L05	Unused State Variable	Unresolved
Minor / Informative	L08	Tautology or Contradiction	Unresolved
Minor / Informative	L11	Unnecessary Boolean equality	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
Audit Updates 5
Source Files 5
Findings Breakdown 6
ST - Stops Transactions 7
Description 7
Recommendation 7
ELFM - Exceeds Fees Limit 8
Description 8
Recommendation 9
SFV - Swap Functionality Vulnerability 10
Description 10
Recommendation 10
RCS - Redundant Conditional Statement 11
Description 11
Recommendation 11
PTRP - Potential Transfer Revert Propagation 12
Description 12
Recommendation 12
FSA - Fixed Swap Address 13
Description 13
Recommendation 13
MEM - Misleading Error Messages 14
Description 14
Recommendation 14
IDI - Immutable Declaration Improvement 15
Description 15
Recommendation 15
L02 - State Variables could be Declared Constant 16
Description 16
Recommendation 16
L04 - Conformance to Solidity Naming Conventions 17
Description 17
Recommendation 18
L05 - Unused State Variable 19
Description 19

Recommendation 19
L08 - Tautology or Contradiction 20
Description 20
Recommendation 20
L11 - Unnecessary Boolean equality 21
Description 21
Recommendation 21
Functions Analysis **22**
Inheritance Graph **29**
Flow Graph **30**
Summary **31**
Disclaimer **32**
About Cyberscope **33**

Review

Contract Name	GCT
Testing Deploy	https://testnet.bscscan.com/address/0xa3d0c4a1770e0a6aab500c9ec9e684302dc7a0af
Symbol	GCT
Decimals	10
Total Supply	10,000,000

Audit Updates

Initial Audit	08 Jun 2023 https://github.com/cyberscope-io/audits/blob/main/v1/2-gct/audit.pdf
Corrected Phase 2	12 Jun 2023

Source Files

Filename	SHA256
contracts/GameCraft.sol	a216f96cb8f1b22a2d88b8ced1e0f2ae79b74bdd94d8c12082cc0d02d00afe45

Findings Breakdown

Critical	3
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	GCT.sol#L681
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(checkWalletLimit && !isWalletLimitExempt[to])
  require(balanceOf(to).add(amount) <= _walletMax);
```

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	contracts/GameCraft.sol#L831,838,845,852
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 following functions with a high percentage value:

- `setRewardFeePercent` .
- `setLiquidityFeePercent` .
- `setDeveloperFeePercent` .
- `setBurnFeePercent` .

```
function setRewardFeePercent(uint256 RewardFee) external onlyOwner() {
    require(RewardFee >= 0 && RewardFee <= 25, "Fee can't be set more than
25%");
    _RewardFee = RewardFee;

    emit UpdateData("_RewardFee", RewardFee);
}
function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner()
{
    require(liquidityFee >= 0 && liquidityFee <= 25, "Fee can't be set
more than 25%");
    _liquidityFee = liquidityFee;

    emit UpdateData("_liquidityFee", liquidityFee);
}
...
```

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.

SFV - Swap Functionality Vulnerability

Criticality	Critical
Location	contracts/GameCraft.sol#L768
Status	Unresolved

Description

As part of the transfer flow, the contract has two implementations of the swap functionality. However, one of them lacks a mutex in its implementation. During the swapping process, the `transfer` function will be called internally. As a consequence of the second swap lacking a mutex, the contract can fall into an infinite loop, continuously executing the swap operation. This continuous execution consumes all the available gas and eventually reaches the limit, causing the transaction to fail and revert.

```
uint256 initialBalance = address(this).balance;

swapTokensForEth(DeveloperAmt);

uint256 newBalance = address(this).balance.sub(initialBalance);
payable(DeveloperWallet).transfer(newBalance);
```

Recommendation

The team is advised to take these segments into consideration and either remove the second `swapTokensForEth` function or introduce a mutex. This way the contract will avoid entering an infinite loop, causing the transaction to revert.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	contracts/GameCraft.sol#L789
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 executes the same check twice. Once at the `_transfer` function, and once at the `_transferStandard` function. As a result, including this check at the `_transferStandard` function is redundant.

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

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/GameCraft.sol#L771
Status	Unresolved

Description

The contract sends funds to a `DeveloperWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(DeveloperWallet).transfer(newBalance);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	GCT.sol#L400
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.

```
IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0xD99D1c33F9fC3444f8101754aBC46c52416550D1);
 // Create a uniswap pair for this new token
uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
    .createPair(address(this), _uniswapV2Router.WETH());

// set the rest of the contract variables
uniswapV2Router = _uniswapV2Router;
```

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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/GameCraft.sol#L636
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(newLimit <= totalSupply().mul(3).div(100),
    "_walletMax can't be set more than more than 10% or lesser than 1%");
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/GameCraft.sol#L402
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
```

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	contracts/GameCraft.sol#L92,93,348,376,379
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.

```
address private _previousOwner
uint256 private _lockTime
uint256 private _tTotal = 10000000 * 10**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	contracts/GameCraft.sol#L159,160,176,197,356,359,362,365,366,369,370,589,595,831,845
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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	contracts/GameCraft.sol#L92,93
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
address private _previousOwner
uint256 private _lockTime
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	contracts/GameCraft.sol#L832,839,846,853
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(RewardFee >= 0 && RewardFee <= 25, "Fee can't be set more than
25%")
require(liquidityFee >= 0 && liquidityFee <= 25, "Fee can't be set more
than 25%")
require(DeveloperFee >= 0 && DeveloperFee <= 25, "Fee can't be set more
than 25%")
require(burnFee >= 0 && burnFee <= 25, "Fee can't be set more than 25%")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/GameCraft.sol#L647,654
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
emit UpdateData("checkWalletLimit", newValue == true ? 1 : 0 )
emit UpdateData("isWalletLimitExempt", exempt == true ? 1 : 0 )
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in 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		
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	✓	-

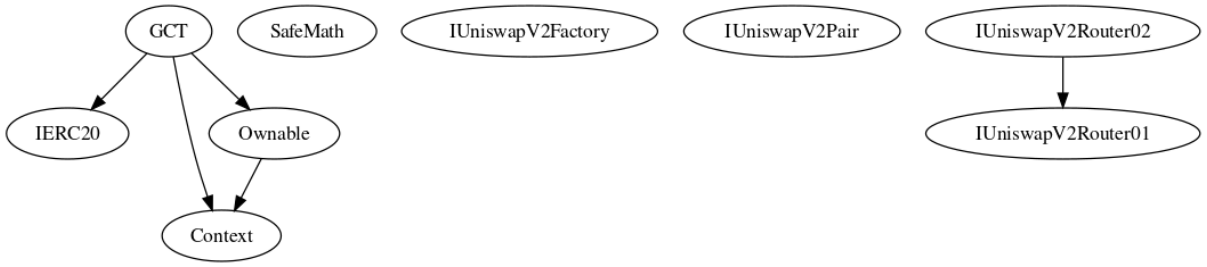
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	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	✓	-
	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	✓	-

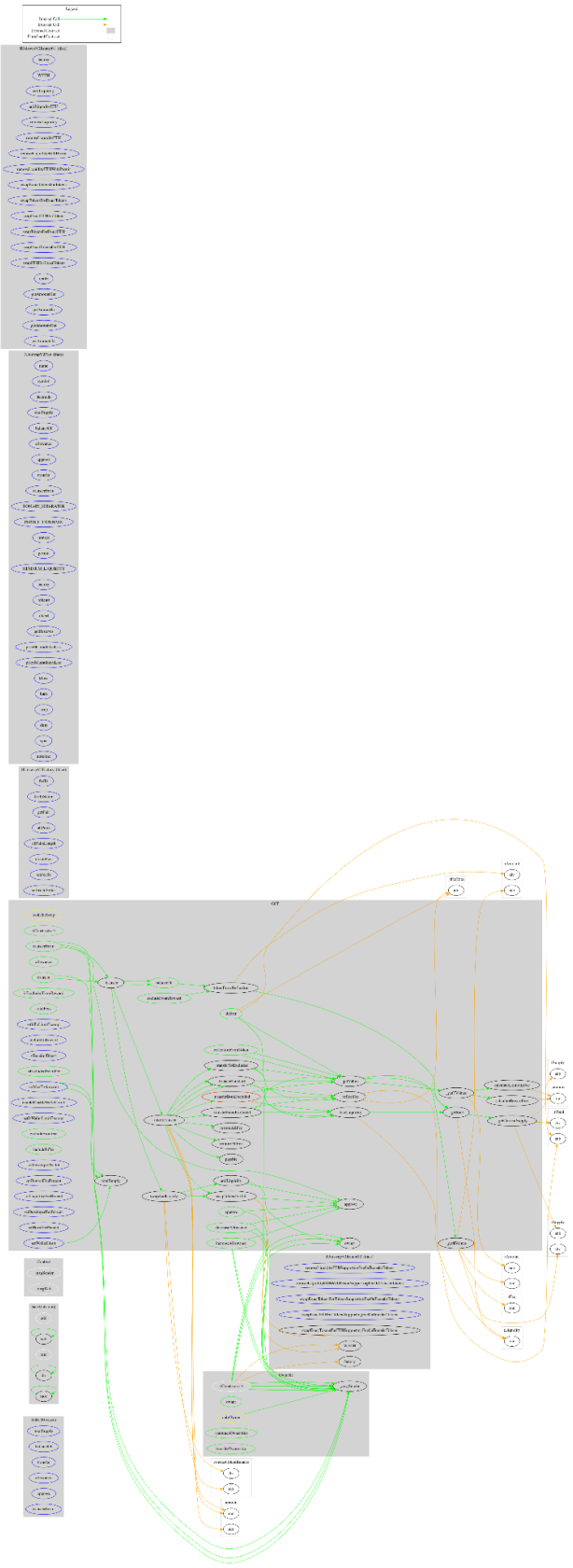
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_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 stopping transactions and manipulating the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Audit Comment

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. According to the previously mentioned issues, the contract cannot be assumed that it is in a production-ready state. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure the contract adheres to established best practices and security measures. It is recommended that the team review the contract's gas consumption and optimize it accordingly to minimize costs and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

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