SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Gamia
GIA
BEP20

0x9CAE159A21a278E0a98Ee42d197aE87Cbc716

Table of Contents

Global Disclaimer

This document serves as a disclaimer for the crypto smart contract audit conducted by Skeleton Ecosystem. The purpose of the audit was to review the codebase of the smart contracts for potential vulnerabilities and issues. It is important to note the following:

Limited Scope: The audit is based on the code and information available up to the audit completion date. It does not cover external factors, system interactions, or changes made after the audit. The audit itself can not guarantee 100% safaty and can not detect common scam methods like farming and developer sell-out.

No Guarantee of Security: While we have taken reasonable steps to identify vulnerabilities, it is impossible to guarantee the complete absence of security risks or issues. The audit report provides an assessment of the contract's security as of the audit date.

Continued Development: Smart contracts and blockchain technology are evolving fields. Updates, forks, or changes to the contract post-audit may introduce new risks that were not present during the audit.

Third-party Code: If the smart contract relies on third-party libraries or code, those components were not thoroughly audited unless explicitly stated. Security of these dependencies is the responsibility of their respective developers.

Non-Exhaustive Testing: The audit involved automated analysis, manual review, and testing under controlled conditions. It is possible that certain vulnerabilities or issues may not have been identified.

Risk Evaluation: The audit report includes a risk assessment for identified vulnerabilities. It is recommended that the development team carefully reviews and addresses these risks to mitigate potential exploits.

Not Financial Advice: This audit report is not intended as financial or investment advice. Decisions regarding the use, deployment, or investment in the smart contract should be made based on a comprehensive assessment of the associated risks.

By accessing and using this audit report, you acknowledge and agree to the limitations outlined above. Skeleton Ecosystem and its auditors shall not be held liable for any direct or indirect damages resulting from the use of the audit report or the smart contract itself.

Please consult with legal, technical, and financial professionals before making any decisions related to the smart contract.

Overview

Contract Name	Gamia
Ticker/Simbol	GIA
Blockchain	Binance Smart Chain BEP20
Contract Address	0x9CAE159A21a278E0a98Ee42d197aE87Cbc7165B3
Creator Address	0x25603d3c271ef7Db66cE5F431eFbB5cE0377BBa5
Current Owner Address	0x25603d3c271ef7Db66cE5F431eFbB5cE0377BBa5
Contract Explorer	https://bscscan.com/token/0x9cae159a21a278e0a98ee42d197ae87cbc7165b3#code
Compiler Version	v0.8.17+commit.8df45f5f
License	None
Optimisation	No with 200 Runs
Total Supply	10,000,000,000 **GIA**
Decimals	9

Creation/Audit

Contract Deployed	10.02.2024
Audit Created	12.02.2024
Audit Update	V 1.0

Verified Socials

Website	https://gamia.net/
Telegram	https://t.me/Gamiaportal
Twitter (X)	https://twitter.com/gamiaplatform

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

Contract Function Analysis

✅ Pass ⚠️ Attention Item 🔺 Risky Item

■ Pass
■ Attention
■ Risk

Contract Verified	✅	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0x25603d3c271ef7Db66cE5F431eFbB5cE0377BBa5 Deployer
Buy Tax	4 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Fee can be set!
Sell Tax	4 %	Shows the taxes for sell transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Fee can be set!
Honeypot Analyse	✅	Holder is able to buy and sell. If honeypot: The contract blocks sell transfer from holder wallet. Multiple events may cause honeypot. Trading disabled, extremely high tax
Liqudity Status		Presale
Trading Disable Functions	✅	No Trading suspendable function found. If a suspendable code is included, the token maybe neither be bought or sold (honeypot risk). If contract is renounced this function can't be used
Set Fees function	✅	No Fee Setting function found. The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk).
Proxy Contract	✅	Not a Proxy contract.
Mint Function	✅	No Mint Function detected Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token. Owner can mint new tokens and sell.

Balance Modifier Function	✅	No Balance Modifier function found. If there is a function for this, the contract owner can have the authority to modify the balance of tokens at other addresses. For example revoke the bought tokens from the holders wallet. Common form of scam: You buy the token, but it's disappearing from your wallet.
Blacklist Function	⚠️	Blacklist Setting function found. Reduce risk: only possible to exclude wallets from receiving dividend. Wallets from trading can not be blacklisted. If there is a blacklist, some addresses may not be able to trade normally. Example: you buy the token and right after your Wallet getting blacklisted. Like so you will be unable to sell. Honeypot Risk.
Whitelist Function	⚠️	Whitelist Setting function found. If there is a function for this, Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee. Can cause farming)
Hidden Owner Analysis	✅	No Hidden or multi owner with authorisation For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned.
Retrieve Ownership Function	✅	No Functions found which can retrieve ownership of the contract. If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.
Self Destruct Function	✅	No Self Destruct function found. If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.
Specific Tax Changing Function	✅	No Specific Tax Changing Functions found. If it exists, the contract owner may set a very outrageous tax rate for assigned address to block it from trading. Can assign all wallets at once!
Trading Cooldown Function	✅	No Trading Cooldown Function found. If there is a trading cooldown function, the user will not be able to sell the token within a certain time or block after buying. Like a temporary honeypot.
Max Transaction and Holding Modify Function	✅	No Max Transaction and Holding Modify function found. Remove all limits function triggered If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot
Transaction Limiting Function	✅	No Transaction Limiter Function Found. The number of overall token transactions may be limited (honeypot risk)

Details of Risk - Attention Items

⚠️ Whitelist function (Exclude from Fees)

If there is a function for this Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee. Can cause farming)

```
ftrace | funcSig
1852    function excludeFromFees(address account, bool excluded) public {
1853        require(msg.sender == SafeManager);
1854        require(_isExcludedFromFees[account] != excluded, "Gamia: Account is already the value of 'excluded'");
1855        _isExcludedFromFees[account] = excluded;
1856
1857        emit ExcludeFromFees(account, excluded);
1858    }
```

⚠️ Blacklist (Exclude from Dividend)

✅ Reduced risk: only possible to exclude wallets from receiving divident. Trading can not be blacklisted. Wallets from trading can not be blacklisted.

```
ftrace | funcSig
1867    function excludeFromDividends(address account) external {
1868        require(msg.sender == SafeManager);
1869        dividendTracker.excludeFromDividends(account);
1870    }
1871
```

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

Contract Security

Total Findings: 7

- High 0
- Medium 1
- Low 5
- Info 1
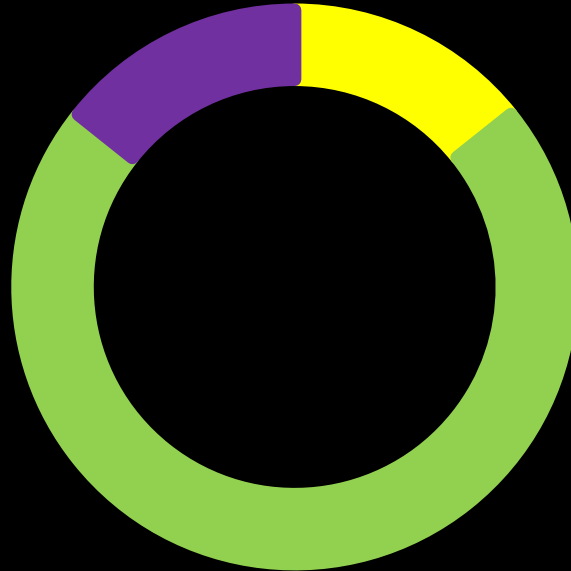
High Severity Issues: High possibility to cause problems, need to be resolved.

Medium Severity Issue: Will likely cause problems, recommended to resolve.

Low Severity Issues: Won't cause problems, but for improvement purposes could be adjusted.

Informational Severity Issues: Not harmful in any way, information for the developer team.

Contract Security

List of Found Issues

■ **High severity Issues: (0)**

■ **Medium severity issues: (1)**

- Authorization through tx.origin

■ **Low severity issues: (4)**

- Long number literals
- Approve Front Running Attack (Sandwich Bots)
- Low Level Calls
- Public Functions Should be Declared External
- Reentrancy

■ **Informational severity issues: (1)**

- Public Functions Should be Declared External

Contract Weakness Classisication

THE SMART CONTRACT WEAKNESS CLASSIFICATION REGISTRY (SWC REGISTRY) IS AN IMPLEMENTATION OF THE WEAKNESS CLASSIFICATION SCHEME PROPOSED IN EIP-1470. IT IS LOOSELY ALIGNED TO THE TERMINOLOGIES AND STRUCTURE USED IN THE COMMON WEAKNESS ENUMERATION (CWE) WHILE OVERLAYING A WIDE RANGE OF WEAKNESS VARIANTS THAT ARE

ID	Description	AI	Manual	Result
SWC-100	Function Default Visibility	Passed	Passed	Passed
SWC-101	Integer Overflow and Underflow	Passed	Passed	Passed
SWC-102	Outdated Compiler Version	Passed	Passed	Passed
SWC-103	Floating Pragma	Passed	Passed	Passed
SWC-104	Unchecked Call Return Value	Passed	Passed	Passed
SWC-105	Unprotected Ether Withdrawal	Passed	Passed	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed	Passed	Passed
SWC-107	Reentrancy	Passed	Passed	Passed
SWC-108	State Variable Default Visibility	Passed	Passed	Passed
SWC-109	Uninitialized Storage Pointer	Passed	Passed	Passed
SWC-110	Assert Violation	Passed	Passed	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed	Passed	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed	Passed	Passed
SWC-113	DoS with Failed Call	Passed	Passed	Passed
SWC-114	Transaction Order Dependence	Passed	Passed	Passed
SWC-115	Authorization through tx.origin	High	Medium	Medium
SWC-116	Block values as a proxy for time	Passed	Passed	Passed
SWC-117	Signature Malleability	Passed	Passed	Passed
SWC-118	Incorrect Constructor Name	Passed	Passed	Passed

SWC-119	Shadowing State Variables	Passed	Passed	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed	Passed	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed	Passed	Passed
SWC-122	Lack of Proper Signature Verification	Passed	Passed	Passed
SWC-123	Requirement Violation	Passed	Passed	Passed
SWC-124	Write to Arbitrary Storage Location	Passed	Passed	Passed
SWC-125	Incorrect Inheritance Order	Passed	Passed	Passed
SWC-126	Insufficient Gas Griefing	Passed	Passed	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed	Passed	Passed
SWC-128	DoS With Block Gas Limit	Passed	Passed	Passed
SWC-129	Typographical Error	low	Passed	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed	Passed	Passed
SWC-131	Presence of unused variables	Passed	Passed	Passed
SWC-132	Unexpected Ether balance	Passed	Passed	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed	Passed	Passed
SWC-134	Message call with hardcoded gas amount	Passed	Passed	Passed
SWC-135	Code With No Effects	Passed	Passed	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed	Passed	Passed

Detected High and Medium Severity Vulnerability Description.

⚠️ Authorization through tx.origin (2 Items)

Item: 1	Location:	Line 1972	Severity:	🟨 Medium
Item: 2	Location:	Line 2029	Severity:	🟨 Medium

Function	In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable. For example, if an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.
Remedation	tx.origin should not be used for authorization in smart contracts. It does have some legitimate use cases, for example, To prevent external contracts from calling the current contract, you can implement a require of the form require(tx.origin == msg.sender). This prevents intermediate contracts from calling the current contract, thus limiting the contract to regular codeless addresses.

```
      ftrace | funcSig
1970    function processDividendTracker(uint256 gas!) external {
1971        (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) = dividendTracker.process(gas!);
1972        emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, false, gas!, tx.origin);
1973    }
```

```
2027
2028    try dividendTracker.process(gas) returns (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) {
2029        emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, true, gas, tx.origin);
2030    }
```

⚠️ Reentrancy (1 Item)

Item: 1	Location:	Line 1889-1895	Severity:	🟩 Low

Function	In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls. This may lead to loss of funds, improper value updates, token loss, etc.
Remediation	It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

```
ftrace | funcSig
1889    function setAutomatedMarketMakerPairInitial() external {
1890        require(msg.sender == SafeManager);
⚠ 1891        address _uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory()).getPair(address(this), uniswapV2Router.WETH());
1892
1893        uniswapV2Pair = _uniswapV2Pair;
1894        _setAutomatedMarketMakerPair(_uniswapV2Pair, true);
1895    }
1896
```

⚠️ Approve of front running attack. Also known as Sandwich Bot attack. (2 Item)

Item: 1	Location:	Line 923-926	Severity:	🟩 Low

Function	The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function approve can be front-run by abusing the _approve function.
Remedation	1.Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions. 2.Use transaction taxes to prevent against front-runattack

```
trace | func.sig
923    function approve(address spender, uint256 amount) public virtual override returns (bool) {
924        _approve(_msgSender(), spender, amount);
925        return true;
926    }
```

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

Item: 2	Location:	Line 914-955	Severity:	■ Low

Function	The transferFrom() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function transferFrom can be front-run by abusing the _approve function.
Remedation	1.Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions. 2.Use transaction taxes to prevent against front-runattack

```
ftrace | funcSig
941     function transferFrom(
942         address sender,
943         address recipient,
944         uint256 amount
945     ) public virtual override returns (bool) {
946         _transfer(sender, recipient, amount);
947
948         uint256 currentAllowance = _allowances[sender][_msgSender()];
949         require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
950         unchecked {
951             _approve(sender, _msgSender(), currentAllowance - amount);
952         }
953
954         return true;
955     }
```

Contract Flow Graph
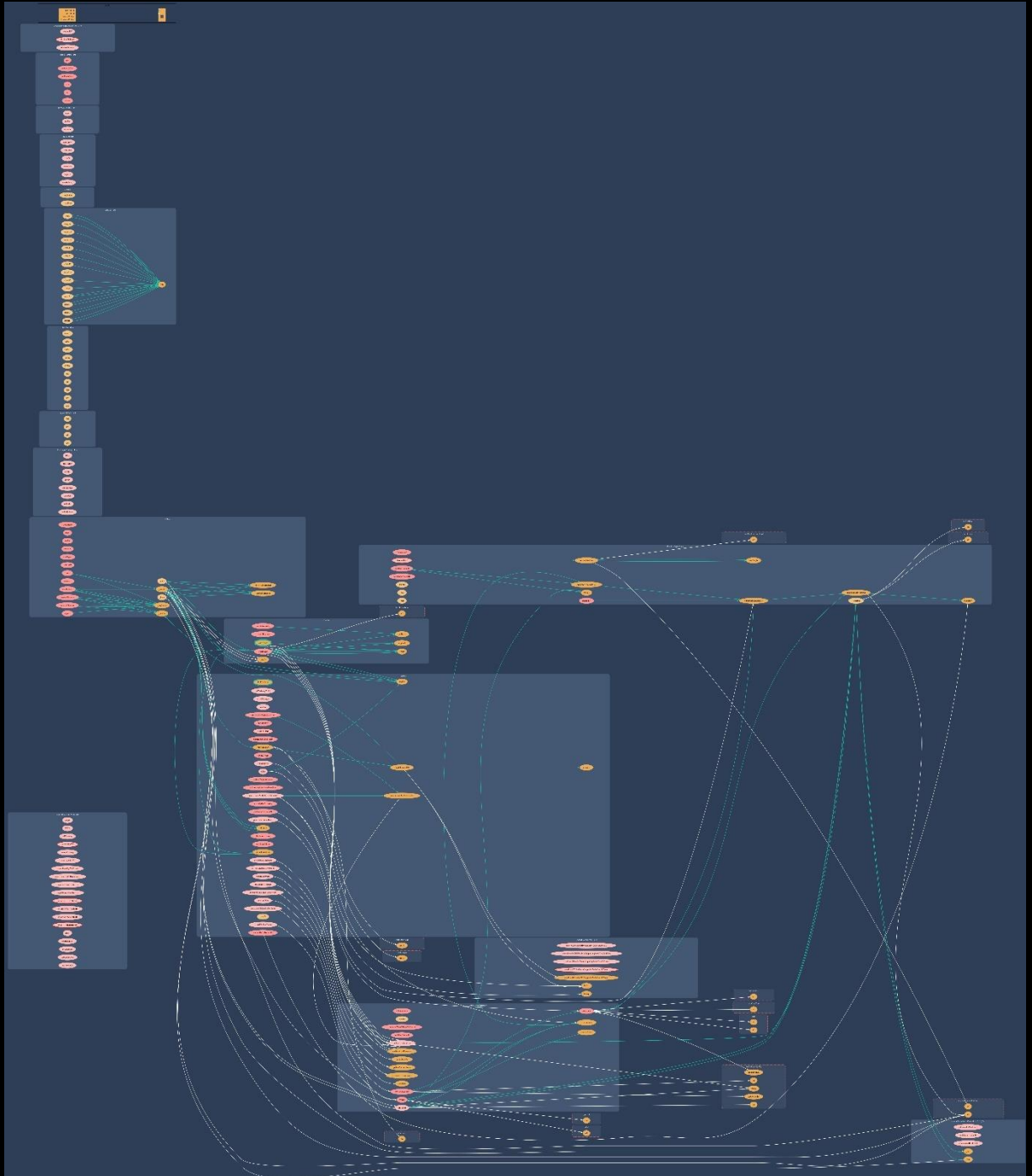
Contract Interaction Graph

Inheritance Graph

IUniswapV2Router02

IUniswapV2Factory

SignedSafeMath

SafeMath

SafeCast

GamiaDividendTracker

IterableMapping

IUniswapV2Router01

Gamia

DividendPayingToken

Ownable

ERC20

DividendPayingTokenOptionalInterface

DividendPayingTokenInterface

Context

IERC20Metadata

IERC20
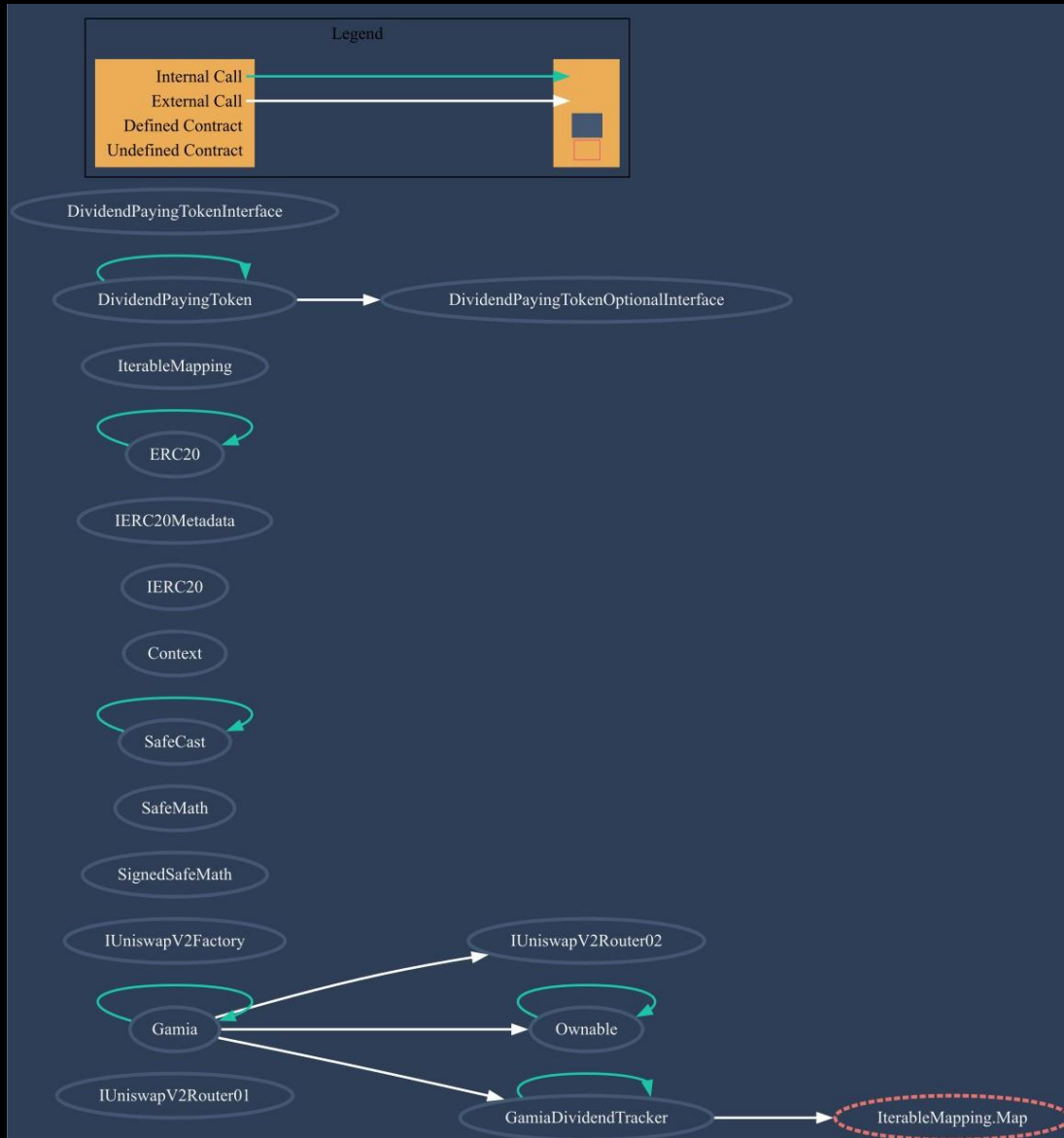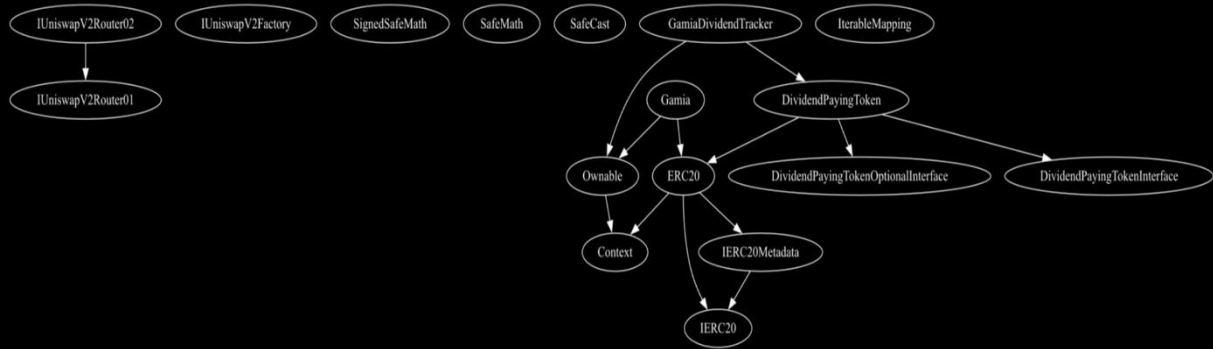
Contract Functions

Contract	Type	Bases		
∟	Function Name	Visibility	Mutability	Modifiers
IUniswapV2Router01	Interface			
∟	factory	External		NO
∟	WETH	External		NO
∟	addLiquidity	External	◯	NO
∟	addLiquidityETH	External	💰	NO
∟	removeLiquidity	External	◯	NO
∟	removeLiquidityETH	External	◯	NO
∟	removeLiquidityWithPermit	External	◯	NO
∟	removeLiquidityETHWithPermit	External	◯	NO
∟	swapExactTokensForTokens	External	◯	NO
∟	swapTokensForExactTokens	External	◯	NO
∟	swapExactETHForTokens	External	💰	NO
∟	swapTokensForExactETH	External	◯	NO
∟	swapExactTokensForETH	External	◯	NO
∟	swapETHForExactTokens	External	💰	NO

	quote	External		NO
L	quote	External		NO
L	getAmountOut	External		NO
L	getAmountIn	External		NO
L	getAmountsOut	External		NO
L	getAmountsIn	External		NO
IUniswapV2Router02	Interface	IUniswapV2Router01		
L	removeLiquidityETHSupportingFeeOnTransferTokens	External	◉	NO
L	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	◉	NO
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	◉	NO
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External	💳	NO
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External	◉	NO
IUniswapV2Factory	Interface			
L	feeTo	External		NO
L	feeToSetter	External		NO

L	getPair	External 🛡		NO🛡
L	allPairs	External 🛡		NO🛡
L	allPairsLength	External 🛡		NO🛡
L	createPair	External 🛡	⬡	NO🛡
L	setFeeTo	External 🛡	⬡	NO🛡
L	setFeeToSetter	External 🛡	⬡	NO🛡
SignedSafeMath	Library			
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	sub	Internal 🔒		
L	add	Internal 🔒		
SafeMath	Library			
L	tryAdd	Internal 🔒		
L	trySub	Internal 🔒		
L	tryMul	Internal 🔒		
L	tryDiv	Internal 🔒		
L	tryMod	Internal 🔒		
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		

∟	sub	Internal 🔒		
∟	div	Internal 🔒		
∟	mod	Internal 🔒		
SafeCast	Library			
∟	toUint224	Internal 🔒		
∟	toUint128	Internal 🔒		
∟	toUint96	Internal 🔒		
∟	toUint64	Internal 🔒		
∟	toUint32	Internal 🔒		
∟	toUint16	Internal 🔒		
∟	toUint8	Internal 🔒		
∟	toUint256	Internal 🔒		
∟	toInt128	Internal 🔒		
∟	toInt64	Internal 🔒		
∟	toInt32	Internal 🔒		
∟	toInt16	Internal 🔒		
∟	toInt8	Internal 🔒		
∟	toInt256	Internal 🔒		
Context	Implementation			
∟	_msgSender	Internal 🔒		
∟	_msgData	Internal 🔒		
IERC20	Interface			

	totalSupply	External		NO
L	totalSupply	External		NO
L	balanceOf	External		NO
L	transfer	External	◉	NO
L	allowance	External		NO
L	approve	External	◉	NO
L	transferFrom	External	◉	NO
IERC20Metadata	Interface	IERC20		
L	name	External		NO
L	symbol	External		NO
L	decimals	External		NO
ERC20	Implementation	Context, IERC20, IERC20Metadata		
L		Public	◉	NO
L	name	Public		NO
L	symbol	Public		NO
L	decimals	Public		NO
L	totalSupply	Public		NO
L	balanceOf	Public		NO
L	transfer	Public	◉	NO
L	allowance	Public		NO
L	approve	Public	◉	NO
L	transferFrom	Public	◉	NO

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

L	increaseAllowance	Public 🛡	⬢	NO🛡
L	decreaseAllowance	Public 🛡	⬢	NO🛡
L	_transfer	Internal 🔒	⬢	
L	_mint	Internal 🔒	⬢	
L	_burn	Internal 🔒	⬢	
L	_approve	Internal 🔒	⬢	
L	_beforeTokenTransfer	Internal 🔒	⬢	
L	_afterTokenTransfer	Internal 🔒	⬢	
Ownable	Implementation	Context		
L		Public 🛡	⬢	NO🛡
L	owner	Public 🛡		NO🛡
L	renounceOwnership	Public 🛡	⬢	onlyOwner
L	transferOwnership	Public 🛡	⬢	onlyOwner
L	_setOwner	Private 🔏	⬢	
IterableMapping	Library			
L	get	Public 🛡		NO🛡
L	getIndexOfKey	Public 🛡		NO🛡
L	getKeyAtIndex	Public 🛡		NO🛡
L	size	Public 🛡		NO🛡

L	set	Public 🛢	⬡	NO🛢
L	remove	Public 🛢	⬡	NO🛢
DividendPayingTokenOptionalInterface	Interface			
L	withdrawableDividendOf	External 🛢		NO🛢
L	withdrawnDividendOf	External 🛢		NO🛢
L	accumulativeDividendOf	External 🛢		NO🛢
DividendPayingTokenInterface	Interface			
L	dividendOf	External 🛢		NO🛢
L	distributeDividends	External 🛢	💴🛢	NO🛢
L	withdrawDividend	External 🛢	⬡	NO🛢
DividendPayingToken	Implementation	ERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
L		Public 🛢	⬡	ERC20
L		External 🛢	💴🛢	NO🛢
L	distributeDividends	Public 🛢	💴🛢	NO🛢
L	withdrawDividend	Public 🛢	⬡	NO🛢

L	_withdrawDividendOfUser	Internal 🔒	⬟	
L	dividendOf	Public ▯		NO▯
L	withdrawableDividendOf	Public ▯		NO▯
L	withdrawnDividendOf	Public ▯		NO▯
L	accumulativeDividendOf	Public ▯		NO▯
L	_transfer	Internal 🔒	⬟	
L	_mint	Internal 🔒	⬟	
L	_burn	Internal 🔒	⬟	
L	_setBalance	Internal 🔒	⬟	
GamiaDividendTracker	Implementation	DividendPaying Token, Ownable		
L		Public ▯	⬟	DividendPaying Token
L	_transfer	Internal 🔒		
L	_minimumTokenBalanceForReward	Public ▯	⬟	onlyOwner
L	withdrawDividend	Public ▯		NO▯
L	excludeFromDividends	External ▯	⬟	onlyOwner
L	updateClaimWait	External ▯	⬟	onlyOwner
L	getLastProcessedIndex	External ▯		NO▯

L	getNumberOfTokenHolders	External 🔳		NO🔳
L	getAccount	Public 🔳		NO🔳
L	getAccountAtIndex	Public 🔳		NO🔳
L	canAutoClaim	Private 🍞		
L	setBalance	External 🔳	⬡	onlyOwner
L	process	Public 🔳	⬡	NO🔳
L	processAccount	Public 🔳	⬡	onlyOwner
Gamia	Implementation	ERC20, Ownable		
L	setMarketingWallet	External 🔳	⬡	NO🔳
L	setSafeManager	External 🔳	⬡	NO🔳
L	withdraw	External 🔳	⬡	NO🔳
L	withdrawBNB	External 🔳	⬡	NO🔳
L		Public 🔳	⬡	ERC20
L		External 🔳	🔳🔳	NO🔳
L	updateUniswapV2Router	Public 🔳	⬡	NO🔳
L	excludeFromFees	Public 🔳	⬡	NO🔳
L	includeToFees	Public 🔳	⬡	NO🔳
L	excludeFromDividends	External 🔳	⬡	NO🔳
L	setAutomatedMarketMakerPair	Public 🔳	⬡	NO🔳

L	setSwapTokensAtAmount	Public ▯	⬡	NO▯
L	setExpectingBnbAmountFromSwap	Public ▯	⬡	NO▯
L	setAutomatedMarketMakerPairInitial	External ▯	⬡	NO▯
L	_setAutomatedMarketMakerPair	Private 🔏	⬡	
L	updateGasForProcessing	Public ▯	⬡	NO▯
L	updateClaimWait	External ▯	⬡	onlyOwner
L	getClaimWait	External ▯		NO▯
L	getTotalDividendsDistributed	External ▯		NO▯
L	isExcludedFromFees	Public ▯		NO▯
L	isIncludedToFees	Public ▯		NO▯
L	withdrawableDividendOf	Public ▯		NO▯
L	dividendTokenBalanceOf	Public ▯		NO▯
L	getAccountDividendsInfo	External ▯		NO▯
L	getAccountDividendsInfoAtIndex	External ▯		NO▯
L	processDividendTracker	External ▯	⬡	NO▯

L	claim	External 〗	◉	NO〗
L	getLastProcessedIndex	External 〗	⋈	NO〗
L	getNumberOfDividendTokenHolders	External 〗		NO〗
L	_transfer	Internal 🔒	◉	
L	swapAndLiquify	Private 🍞	◉	lockTheSwap
L	swapTokensForBnb	Private 🍞	◉	

◉ Function can modify state

💵 Function is payable

Audit Scope

Audit Method.

Our smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. Goal: discover errors, issues and security vulnaribilities in the code. Findings getting reported and improvements getting suggested.

Automatic and Manual Review
We are using automated tools to scan functions and weeknesses of the contract. Transfers, integer over-undeflow checks such as all CWE events.

Tools we use:
Visual Studio Code
CWE
SWC
Solidity Scan
SVD

In manual code review our auditor looking at source code and performing line by line examination. This method helps to clarify developer's coding decisions and business logic.

Skeleton Ecosystem

https://skeletonecosystem.com

https://github.com/SkeletonEcosystem/Audits