



Grok Inu Contract Code Review and Security Analysis Report

Customer: Grok Inu
Prepared on: 28th December 2023
Platform: Binance Smart Chain
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	2
Document	3
Introduction	4
Project Scope	5
Executive Summary	6
Code Quality	6
Documentation	8
Use of Dependencies	8
AS-IS Overview	9
Code Flow Diagram - Grok Inu	16
Interaction Diagram	18
Inheritance Diagram	19
Code Flow Diagram - Slither Results Log	20
Audit Findings	27
Conclusion	32
Note For Contract Users	32
Our Methodology	40
Disclaimers	42

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of Grok Inu
Platform	Binance Smart Chain/ Solidity
File	GrokInu.sol
MD5 hash	4ee498bf4152089aeffeefb8a14548b7
SHA256 hash	6f002c042ea8cdfef9de941b5c840d11c5bc46de196fb60d93f4cbb0194f178f
Date	28/12/2023

Introduction

RD Auditors (Consultant) were contracted by GrokInu (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer's smart contract and its code review conducted between 23rd - 28th December 2023.

This contract consists of one file.

Project Scope





The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary






According to the assessment, the customer's solidity smart contract is now **Secured**.

You are Here

 **Insecure**  **Poorly Secured**  **Secure**  **Well-Secured**

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	2
 Critical	0
 High	0
 Medium	0
 Low	2
 Very Low	0

Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The Grok Inu team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the Grok Inu code as a link:

<https://bscscan.com/token/0xd08b9e557d1f64c8dd50a168453ea302a83e47fc>

The hash of that file is mentioned in the table. As mentioned above, it's commented on smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

GrokInu.sol

File And Function Level Report

Contract: ERC20
Inherit: Context, IERC20, IERC20Metadata
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	IncreaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	_transfer	internal	Passed	All Passed	No Issue	Passed
13	_mint	internal	Passed	All Passed	No Issue	Passed
14	_burn	internal	Passed	All Passed	No Issue	Passed

15	_approve	internal	Passed	All Passed	No Issue	Passed
16	_spendAllowance	internal	Passed	All Passed	No Issue	Passed
17	_beforeTokenTransfer	internal	Passed	All Passed	No Issue	Passed
18	_afterTokenTransfer	internal	Passed	All Passed	No Issue	Passed

Contract: Ownable
 Inherit: Context
 Observation: Passed
 Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	_CheckOwner	internal	Passed	All Passed	No Issue	Passed
3	renounceOwnership	onlyOwner	Passed	All Passed	No Issue	Passed
4	transferOwnership	onlyOwner	Passed	All Passed	No Issue	Passed
5	_transferOwnership	Internal	Passed	All Passed	No Issue	Passed

Contract: DividendPayingToken
 Inherit: Ownable, ERC20, DividendPayingTokenInterface, ReentrancyGuard
 Observation: Passed
 Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	distributeDividends	onlyowner	Passed	All Passed	No Issue	Passed
2	withdrawDividends	write	Passed	All Passed	No Issue	Passed
3	_withdrawDividendOfUser	internal	Passed	All Passed	No Issue	Passed
4	dividendof	read	Passed	All Passed	No Issue	Passed
5	withdrawableDividendOf	read	Passed	All Passed	No Issue	Passed
6	withdrawDividendOf	read	Passed	All Passed	No Issue	Passed
7	accumulativeDividendOf	read	Passed	All Passed	No Issue	Passed
8	_transfer	internal	Passed	All Passed	No Issue	Passed
9	_mint	internal	Passed	All Passed	No Issue	Passed
10	_burn	internal	Passed	All Passed	No Issue	Passed
11	_setBalance	internal	Passed	All Passed	No Issue	Passed

Contract: DividendTracker

Inherit: Ownable, ERC20, DividendPayingToken

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_transfer	internal	Passed	All Passed	No Issue	Passed
2	withdrawDividend	read	Passed	All Passed	No Issue	Passed

3	updateMinimumTokenBalanceForDividends	onlyOwner	Passed	All Passed	No Issue	Passed
4	updateMinimumDividendForAutoclaim	onlyOwner	Passed	All Passed	No Issue	Passed
5	excludeFromDividends	onlyOwner	Passed	All Passed	No Issue	Passed
6	updateClaimWait	onlyOwner	Passed	All Passed	No Issue	Passed
7	SetLastProcessedIndex	onlyOwner	Passed	All Passed	No Issue	Passed
8	getLastProcessedIndex	read	Passed	All Passed	No Issue	Passed
9	getNumberOfTokenHolders	read	Passed	All Passed	No Issue	Passed
10	getAccount	read	Passed	All Passed	No Issue	Passed
11	getAccountIndex	read	Passed	All Passed	No Issue	Passed
12	canAutoClaim	read	Passed	All Passed	No Issue	Passed
13	setBalance	onlyOwner	Passed	All Passed	No Issue	Passed
14	Process	write	Passed	All Passed	No Issue	Passed
15	ProcessAccount	onlyOwner	Passed	All Passed	No Issue	Passed

Contract: GrokInu
 Inherit: Ownable, ERC20, ReentrancyGuard
 Observation: Passed
 Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	isContract	internal	Passed	All Passed	No Issue	Passed

2	getRouterAddresses	read	Passed	All Passed	No Issue	Passed
3	ClaimStuckTokens	onlyOwner	Passed	All Passed	No Issue	Passed
4	SetMarketingWallet1	onlyOwner	Passed	All Passed	No Issue	Passed
5	SetMarketingWallet2	onlyOwner	Passed	All Passed	No Issue	Passed
6	SetMarketingWallet3	onlyOwner	Passed	All Passed	No Issue	Passed
7	SetSwapTokensAtAmount	onlyOwner	Passed	All Passed	No Issue	Passed
8	toggleSwapBack	onlyOwner	Passed	All Passed	No Issue	Passed
9	toggleAutoDistribute	onlyOwner	Passed	All Passed	No Issue	Passed
10	SetExcludeFromFees	onlyOwner	Passed	All Passed	No Issue	Passed
11	isExcludedFromFees	read	Passed	All Passed	No Issue	Passed
12	setAutomatedMarketMakerPair	onlyOwner	Passed	All Passed	No Issue	Passed
13	isAutomatedMarketMakerPair	read	Passed	All Passed	No Issue	Passed
14	_transfer	internal	Passed	All Passed	No Issue	Passed
15	ProcessReflection	internal	Passed	All Passed	No Issue	Passed
16	SwapBack	internal	Passed	All Passed	No Issue	Passed
17	SwapAndLiquify	write	Passed	All Passed	No Issue	Passed
18	sendBNB	internal	Passed	All Passed	No Issue	Passed
19	ManualSwapBack	write	Passed	All Passed	No Issue	Passed
20	updateBuyFees	onlyOwner	Passed	All Passed	No Issue	Passed
21	updateSellFees	onlyOwner	Passed	All Passed	No Issue	Passed

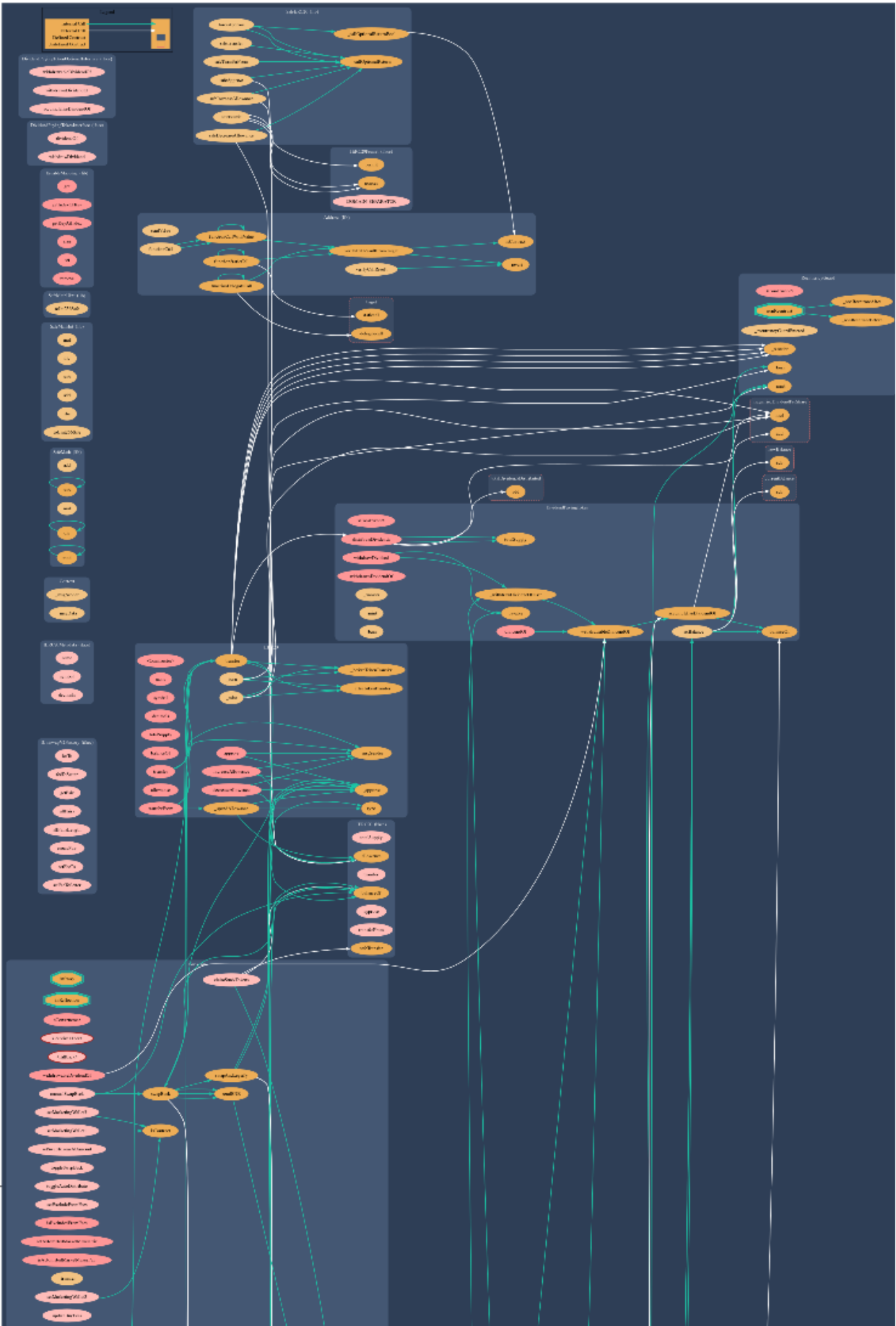
22	updateTransferFees	onlyOwner	Passed	All Passed	No Issue	Passed
23	updatePercentMarketingWallet	onlyOwner	Passed	All Passed	No Issue	Passed
24	UpdateDividendTracker	onlyOwner	Passed	All Passed	No Issue	Passed
25	UpdateGasForProcessing	onlyOwner	Passed	All Passed	No Issue	Passed
26	UpdateMinimumBalanceOfDividends	onlyOwner	Passed	All Passed	No Issue	Passed
27	getMinimumTokenBalanceForDividends	read	Passed	All Passed	No Issue	Passed
28	updateMinimumDividendForAutoClaim	onlyOwner	Passed	All Passed	No Issue	Passed
29	getMinimumDividendForAutoClaim	read	Passed	All Passed	No Issue	Passed
30	updateClaimWait	onlyOwner	Passed	All Passed	No Issue	Passed
31	getClaimWait	read	Passed	All Passed	No Issue	Passed
32	getTotalDividendsDistributed	read	Passed	All Passed	No Issue	Passed
33	withdrawableDividendOf	read	Passed	All Passed	No Issue	Passed
34	dividendTokenBalanceOf	read	Passed	All Passed	No Issue	Passed
35	totalRewardsEarned	read	Passed	All Passed	No Issue	Passed
36	excludeFromDividends	onlyOwner	Passed	All Passed	No Issue	Passed
37	getAccountDividendsInfo	read	Passed	All Passed	No Issue	Passed

38	getAccountDividendsInfoAtIndex	read	Passed	All Passed	No Issue	Passed
39	ProcessDividendTracker	external	Passed	All Passed	No Issue	Passed
40	Claim	onlyOwner	Passed	All Passed	No Issue	Passed
41	ClaimAddress	onlyOwner	Passed	All Passed	No Issue	Passed
42	getLastProcessIndex	read	Passed	All Passed	No Issue	Passed
43	setLastProcessIndex	onlyOwner	Passed	All Passed	No Issue	Passed
44	getNumberOfDividendTokenHolders	read	Passed	All Passed	No Issue	Passed

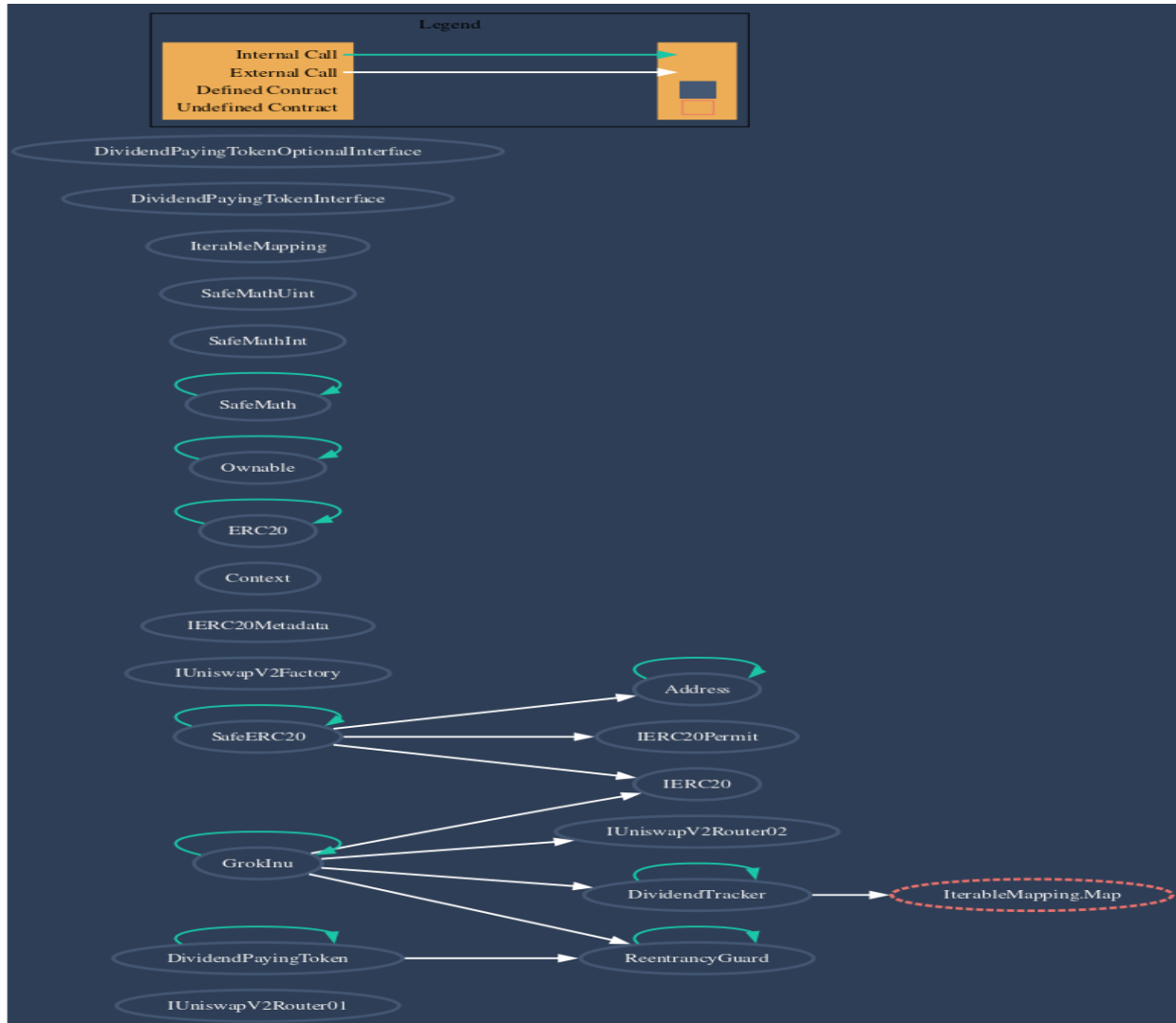
Code Flow Diagram - Grok Inu

More in-depth visuals of the below images are found in this url:

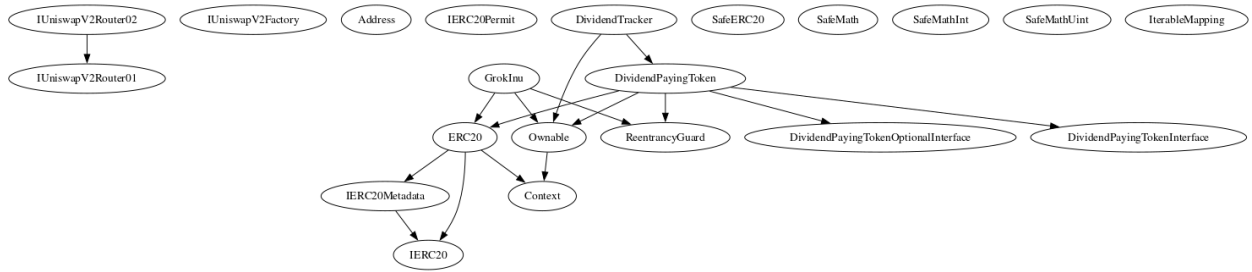
<https://drive.google.com/file/d/1K0CKMthuVST4miShfxA-o9Jsk7FHDLNLO/view?usp=sharing>



Interaction Diagram



Inheritance Diagram



Code Flow Diagram - Slither Results Log

```
INFO:Detectors:
GrokInu.sendBNB(address,uint256) (grokInu.sol#2037-2046) sends eth to arbitrary user
  Dangerous calls:
    - (success) = address(_to).call{value: amount}{} (grokInu.sol#2043)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
```

```
INFO:Detectors:
Reentrancy in GrokInu._transfer(address,address,uint256) (grokInu.sol#1833-1924):
  External calls:
    - swapBack() (grokInu.sol#1864)
      - (success) = address(_to).call{value: amount}{} (grokInu.sol#2043)
      - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,pathToBNB,address(this),block.timestamp) (grokInu.sol#2013-2021)
      - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(contractTokenBalance,0,path,address(this),block.timestamp) (grokInu.sol#1978-1986)
      - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,address(0),block.timestamp) (grokInu.sol#2025-2034)
    External calls sending eth:
      - swapBack() (grokInu.sol#1864)
        - (success) = address(_to).call{value: amount}{} (grokInu.sol#2043)
        - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,address(0),block.timestamp) (grokInu.sol#2025-2034)
  State variables written after the call(s):
    - super._transfer(from,address(this),contractTax) (grokInu.sol#1901)
      - _balances[from] = fromBalance - amount (grokInu.sol#763)
      - _balances[to] += amount (grokInu.sol#765)
  ERC20._balances (grokInu.sol#649) can be used in cross function reentrancies:
    - ERC20._mint(address,uint256) (grokInu.sol#773-785)
    - ERC20._transfer(address,address,uint256) (grokInu.sol#747-771)
    - ERC20.balanceOf(address) (grokInu.sol#679-683)
    - super._transfer(from,address(dividendTracker),reflectionTax) (grokInu.sol#1906)
      - _balances[from] = fromBalance - amount (grokInu.sol#763)
      - _balances[to] += amount (grokInu.sol#765)
  ERC20._balances (grokInu.sol#649) can be used in cross function reentrancies:
    - ERC20._mint(address,uint256) (grokInu.sol#773-785)
    - ERC20._transfer(address,address,uint256) (grokInu.sol#747-771)
```

```
INFO:Detectors:
Contract locking ether found:
  Contract DividendTracker (grokInu.sol#1236-1516) has payable functions:
    - DividendTracker.receive() (grokInu.sol#1270)
    - DividendTracker.fallback() (grokInu.sol#1272)
  But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

```
INFO:Detectors:
Contract locking ether found:
  Contract DividendTracker (grokInu.sol#1236-1516) has payable functions:
    - DividendTracker.receive() (grokInu.sol#1270)
    - DividendTracker.fallback() (grokInu.sol#1272)
  But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

```
INFO:Detectors:
Reentrancy in DividendPayingToken._withdrawDividendOfUser(address) (grokInu.sol#1137-1161):
  External calls:
  - success = IERC20(rewardToken).transfer(user,_withdrawableDividend) (grokInu.sol#1146-1149)
  State variables written after the call(s):
  - withdrawnDividends[user] = withdrawnDividends[user].sub(_withdrawableDividend) (grokInu.sol#1152-1154)
  DividendPayingToken.withdrawnDividends (grokInu.sol#1108) can be used in cross function reentrancies:
  - DividendPayingToken._withdrawDividendOfUser(address) (grokInu.sol#1137-1161)
  - DividendPayingToken.withdrawableDividendOf(address) (grokInu.sol#1167-1171)
  - DividendPayingToken.withdrawnDividendOf(address) (grokInu.sol#1173-1177)
Reentrancy in DividendTracker.process(uint256) (grokInu.sol#1452-1500):
  External calls:
  - processAccount(address(account),true) (grokInu.sol#1481)
    - success = IERC20(rewardToken).transfer(user,_withdrawableDividend) (grokInu.sol#1146-1149)
  State variables written after the call(s):
  - lastProcessedIndex = _lastProcessedIndex (grokInu.sol#1497)
  DividendTracker.lastProcessedIndex (grokInu.sol#1242) can be used in cross function reentrancies:
  - DividendTracker.getAccount(address) (grokInu.sol#1340-1389)
  - DividendTracker.getLastProcessedIndex() (grokInu.sol#1332-1334)
  - DividendTracker.lastProcessedIndex (grokInu.sol#1242)
  - DividendTracker.process(uint256) (grokInu.sol#1452-1500)
  - DividendTracker.setLastProcessedIndex(uint256) (grokInu.sol#1328-1330)
Reentrancy in GrokInu.updateDividendTracker(address) (grokInu.sol#2148-2180):
  External calls:
  - newDividendTracker.excludeFromDividends(address(newDividendTracker)) (grokInu.sol#2163-2165)
```

```
INFO:Detectors:
DividendPayingToken.constructor(string,string,address)._name (grokInu.sol#1111) shadows:
  - ERC20._name (grokInu.sol#655) (state variable)
DividendPayingToken.constructor(string,string,address)._symbol (grokInu.sol#1112) shadows:
  - ERC20._symbol (grokInu.sol#656) (state variable)
DividendPayingToken.dividendOf(address)._owner (grokInu.sol#1163) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
DividendPayingToken.withdrawableDividendOf(address)._owner (grokInu.sol#1168) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
DividendPayingToken.withdrawnDividendOf(address)._owner (grokInu.sol#1174) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
DividendPayingToken.accumulativeDividendOf(address)._owner (grokInu.sol#1180) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

```
INFO:Detectors:
DividendPayingToken.constructor(string,string,address)._name (grokInu.sol#1111) shadows:
  - ERC20._name (grokInu.sol#655) (state variable)
DividendPayingToken.constructor(string,string,address)._symbol (grokInu.sol#1112) shadows:
  - ERC20._symbol (grokInu.sol#656) (state variable)
DividendPayingToken.dividendOf(address)._owner (grokInu.sol#1163) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
DividendPayingToken.withdrawableDividendOf(address)._owner (grokInu.sol#1168) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
DividendPayingToken.withdrawnDividendOf(address)._owner (grokInu.sol#1174) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
DividendPayingToken.accumulativeDividendOf(address)._owner (grokInu.sol#1180) shadows:
  - Ownable._owner (grokInu.sol#848) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

```
INFO:Detectors:
DividendPayingToken.constructor(string,string,address)._rewardToken (grokInu.sol#1113) lacks a zero-check on :
- rewardToken = _rewardToken (grokInu.sol#1115)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
INFO:Detectors:
Address._revert(bytes,string) (grokInu.sol#400-413) uses assembly
- INLINE ASM (grokInu.sol#406-409)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
ReentrancyGuard._nonReentrantBefore() (grokInu.sol#623-627) has costly operations inside a loop:
- _status = _ENTERED (grokInu.sol#626)
ReentrancyGuard._nonReentrantAfter() (grokInu.sol#629-631) has costly operations inside a loop:
- _status = _NOT_ENTERED (grokInu.sol#630)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
GrokInu._transfer(address,address,uint256) (grokInu.sol#1833-1924) has a high cyclomatic complexity (12).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-complexity
INFO:Detectors:
Address.functionCall(address,bytes) (grokInu.sol#261-272) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (grokInu.sol#282-294) is never used and should be removed
Address.functionDelegateCall(address,bytes) (grokInu.sol#345-355) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (grokInu.sol#357-370) is never used and should be removed
Address.functionStaticCall(address,bytes) (grokInu.sol#318-328) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (grokInu.sol#330-343) is never used and should be removed
Address.sendValue(address,uint256) (grokInu.sol#248-259) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (grokInu.sol#388-398) is never used and should be removed
Context._msgData() (grokInu.sol#643-645) is never used and should be removed
DividendPayingToken._transfer(address,address,uint256) (grokInu.sol#1190-1205) is never used and should be removed
ReentrancyGuard._reentrancyGuardEntered() (grokInu.sol#633-635) is never used and should be removed
SafeERC20._callOptionalReturnBool(IERC20,bytes) (grokInu.sol#587-596) is never used and should be removed
SafeERC20.forceApprove(IERC20,address,uint256) (grokInu.sol#537-555) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (grokInu.sol#484-497) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (grokInu.sol#515-535) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (grokInu.sol#499-513) is never used and should be removed
SafeERC20.safePermit(IERC20Permit,address,address,uint256,uint256,uint8,bytes32,bytes32) (grokInu.sol#557-574) is never
```

```
INFO:Detectors:
SafeMathInt.MAX_INT256 (grokInu.sol#956) is never used in SafeMathInt (grokInu.sol#954-993)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors:
GrokInu.reflectionToken (grokInu.sol#1539) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

```
INFO:Detectors:
DividendPayingToken._withdrawDividendOfUser(address) (grokInu.sol#1137-1161) has external calls inside a loop: success =
IERC20(rewardToken).transfer(user,_withdrawableDividend) (grokInu.sol#1146-1149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```


Solidity Static Analysis

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 296:8:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 296:8:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

[more](#)

Pos: 406:16:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 1386:61:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 1430:19:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

[more](#)

Pos: 306:54:

Gas costs:

Gas requirement of function DividendPayingToken.name is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 663:8:

Gas costs:

Gas requirement of function GrokInu.allowance is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 694:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

[more](#)

Pos: 1471:16:

Constant/View/Pure functions:

IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 21:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 1769:12:

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

1) This function requires modification to mitigate potential external reentrancy vulnerabilities.

```
1560 |         modifier inSwap() {
1561 |             swapping = true;
1562 |             _;
1563 |             swapping = false;
1564 |         }
1565 |     }
```

The suggested adjustment is as follows:

```
modifier nonReentrant() {
    require(!swapping, "ReentrancyGuard: reentrant call");
    swapping = true;
    _;
    swapping = false;
}
```

2) This function also requires modification to mitigate potential external reentrancy vulnerabilities.

```
1566     modifier inReflection() {  
1567         reflecting = true;  
1568         _;  
1569         reflecting = false;  
1570     }
```

The suggested adjustment is the same as above.

Very Low:

No very low severity vulnerabilities were found.

Discussion

1) Division Precision: The line `(amount).mul(magnitude) / totalSupply()` involves multiplication and division with potentially large numbers. To avoid precision issues, it's recommended to perform multiplication first and then divide. Additionally, using `SafeMath` for these calculations is advisable to prevent overflow or underflow.

```

function distributeDividends(uint256 amount) public onlyOwner {
  if (totalSupply() > 0) {
    if (amount > 0) {
      magnifiedDividendPerShare = magnifiedDividendPerShare.add(
        (amount).mul(magnitude) / totalSupply()
      );
      emit DividendsDistributed(msg.sender, amount);

      totalDividendsDistributed = totalDividendsDistributed.add(
        amount
      );
    }
  }
}

```

2)Checking for Validity: It's good that the function checks if totalSupply() is greater than 0 before proceeding. You may also want to add an additional check to ensure that the contract has sufficient funds (tokens) to distribute as dividends. This ensures that the contract has enough tokens to cover the specified dividend amount.

```

function distributeDividends(uint256 amount) public onlyOwner {
  if (totalSupply() > 0) {
    if (amount > 0) {
      magnifiedDividendPerShare = magnifiedDividendPerShare.add(
        (amount).mul(magnitude) / totalSupply()
      );
      emit DividendsDistributed(msg.sender, amount);

      totalDividendsDistributed = totalDividendsDistributed.add(
        amount
      );
    }
  }
}

```

3) The `_transfer` function is designed to handle the internal logic of transferring tokens within the contract. It calculates a correction factor based on the amount of tokens transferred and adjusts the magnified dividends for both the sender and the receiver. The `require(false);` statement should be removed for the function to function as intended.

```
1190     function _transfer( 23 gas
1191         address from,
1192         address to,
1193         uint256 value
1194     ) internal virtual override {
1195         require(false);
1196
1197         int256 _magCorrection = magnifiedDividendPerShare
1198             .mul(value)
1199             .toInt256Safe();
1200         magnifiedDividendCorrections[from] = magnifiedDividendCorrections[from]
1201             .add(_magCorrection);
1202         magnifiedDividendCorrections[to] = magnifiedDividendCorrections[to].sub(
1203             _magCorrection
1204         );
1205     }
```



```
function distributedDividends(uint256 amount) public onlyOwner { infu
    if (totalSupply() > 0) {
        if (amount > 0) {
            magnifiedDividendPerShare = magnifiedDividendPerShare.add(
                (amount).mul(magnitude) / totalSupply()
            );
            emit DividendsDistributed(msg.sender, amount);

            totalDividendsDistributed = totalDividendsDistributed.add(
                amount
            );
        }
    }
}
```

4. Overpowered owners: The contracts have many functions that can only be called by the owners. Giving too many privileges to the owners via critical functions might put the user's funds at risk if the owners are compromised or if a rug-pulling attack takes place.

5. Hardcoded addresses: The contract was hardcoding addresses in the code. This may represent that those parameters can never be changed or updated unless it's a proxy contract. It is recommended to go through the code to know more about these hardcoded values and its use.

Conclusion

We were given a contract file and have used all possible tests based on the given object. so it is ready for mainnet deployment. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is "**Secured**".

Note For Contract Users

`distributeDividends`: This function allows the owner of the contract to distribute dividends to token holders.

`UpdateMinimumDividendForAutoClaim`: This function provides a way for the contract owner to adjust the minimum dividend amount required for an automatic claim. Adjusting this threshold might be useful to ensure that only significant dividends trigger an automatic claim, preventing unnecessary gas costs for small dividend amounts.

`ExcludeFromDividends`: This function provides a way for the contract owner to exclude specific accounts from receiving dividends, and it updates related data structures

`UpdateClaimWait`: This is designed to update the `claimWait` variable, which likely represents the time interval (in seconds) that users have to wait between claims

`SetLastProcessedIndex`: This function allows the owner of the contract to update the `lastProcessedIndex` variable to a specified value. The purpose and significance of the `lastProcessedIndex` variable would depend on the broader context of the smart contract.

`SetBalance`: This function allows the contract owner to set the balance of a specified account. If the account is not excluded from dividends and its new balance meets the minimum requirement, it updates the balance and token holders map. If the new balance is below the minimum, it sets the balance to zero and removes the account from the token holders map. Finally, it triggers

the processing of the account's dividends with the processAccount function. The onlyOwner modifier ensures that only the contract owner can execute this function.

ProcessAccount: This function allows the contract owner to process the dividend claim for a specific account. It withdraws any available dividends for the account, updates the last claim time, emits a claim event, and returns true if dividends were claimed. If there were no dividends to claim, it returns false.

ClaimStuckTokens: This function allows the contract owner to claim either native tokens (Ether) or ERC-20 tokens that may have been mistakenly sent to the contract. If the provided token address is the zero address, the function transfers the contract's Ether balance to the owner. If the provided token address is an ERC-20 token, the function transfers the entire balance of that token to the owner. The onlyOwner modifier ensures that only the contract owner can invoke this function.

SetMarketingWallet1: This function allows the contract owner to set the first marketing wallet address (marketingWallet2). It performs similar checks as the previous function to ensure that the new address is different from the current one, not the zero address, and not a contract address.

SetMarketingWallet2: This function allows the contract owner to set the second marketing wallet address (marketingWallet2). It performs similar checks as the previous function to ensure that the new address is different from the current one, not the zero address, and not a contract address.

SetMarketingWallet3: This function allows the contract owner to set the third marketing wallet address (marketingWallet3). It performs checks to ensure that the new address is different from the current one, not the zero address, and not a contract address.

SetSwapTokensAtAmount: This function allows the contract owner to set a specific threshold amount (swapTokensAtAmount) at which a swap operation is triggered. It ensures that the amount is different from the current value and meets a minimum requirement.

toggleSwapBack: This function allows the contract owner to toggle (enable or disable) the "swap back" feature. It ensures that the status is only updated if it is different from the current status.

toggleAutoDistribute: This function allows the contract owner to toggle (enable or disable) the auto-distribution feature. It ensures that the status is only updated if it is different from the current status.

SetExcludeFromFees: This function allows the contract owner to set or update the fee exclusion status for a given account address. It ensures that the status is only updated if it is different from the current status.

setAutomatedMarketMakerPair: This function allows the contract owner to set or update the automated market maker pair status for a given pair address. It ensures that the status is only updated if it is different from the current status.

UpdateBuyFees: This function allows the contract owner to update the buy fees associated with liquidity, marketing, and reflection. It performs validations to ensure that the total fees do not exceed a certain limit and that at least one of the fees is different from the current values. If these conditions are met, it updates the individual fee variables for buying and emits an event to log the update

UpdateSellFees: This function allows the contract owner to update the sell fees associated with liquidity, marketing, and reflection. It performs validations to ensure that the total fees do not exceed a certain limit and that at least one of the fees is different from the current values. If these conditions are met, it updates the individual fee variables for selling and emits an event to log the update.

UpdateTransferFees: This function allows the contract owner to update the transfer fees associated with liquidity, marketing, and reflection. It performs validations to ensure that the total fees do not exceed a certain limit and that at least one of the fees is different from the current values. If these conditions are met, it updates the individual fee variables and emits an event to log the update.

UpdatePercentMarketingWallet: This function allows the contract owner to update the percentage distribution for marketing wallets. It performs validations to ensure that the total percentage is 100% and that at least one of the percentages is different from the current values. If these conditions are met, it updates the individual percentage variables and emits an event to log the update. The ability to update these percentages can be useful for adjusting the allocation of funds to different marketing wallets based on changing requirements or strategies.

UpdateDividendTracker: This function allows the contract owner to update the dividend tracker with a new contract address. It performs necessary validations, creates a new instance of the dividend tracker, configures exclusions, emits an event to log the update, and updates the variable if all conditions are satisfied. The ability to update the dividend tracker is useful for adapting to changes or improvements in dividend distribution logic.

UpdateGasForProcessing: The purpose of this function is to allow the owner to dynamically adjust the amount of gas allocated for processing dividends. Processing dividends can involve computations and interactions with other contracts, and the gas allocation ensures that there is enough computational resources to execute these tasks. Adjusting this value may be necessary

based on factors such as network conditions, gas prices, or contract complexity.

UpdateMinimumBalanceOfDividends: The purpose of this function is to allow the owner to dynamically adjust the minimum token balance required for an address to be eligible to receive dividends. Dividends are typically distributed proportionally based on the token balances of eligible addresses. Adjusting the minimum token balance for dividends can impact the set of addresses eligible to receive dividends.

UpdateMinimumDividendForAutoClaim: This function provides a mechanism for the contract owner to update the minimum dividend amount required for automatic claiming, allowing for dynamic adjustments to the auto-claim threshold.

UpdateClaimWait: The purpose of this function is to allow the owner to dynamically adjust the waiting period required between consecutive claims of dividends. By updating the claimWait variable, the owner can control how frequently users are allowed to claim their dividends.

excludeFromDividends: The excludeFromDividends function is designed to exclude a specific address from receiving dividends.

ClaimAddress: The claimAddress function is a part of the contract's owner-only functionality, allowing the contract owner to manually trigger dividend distribution for a specified Ethereum address. It provides a means for the owner to intervene and ensure that dividends are distributed to a particular address on demand.

SetLastProcessedIndex: The purpose of this function is to provide the owner with the ability to manually update the last processed index of the dividend tracker. This index is likely used to keep track of which addresses have received dividends, preventing duplicate distributions.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.



Email: info@rdauditors.com

Website: www.rdauditors.com

