InterFi NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

AGRIVERSE

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	Agriverse
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0x32F738738F238865f8506593c6e2613647eCE478
Blockchain	Binance Smart Chain
Centralization	Active ownership
Commit	565ddc5b5ccec595f3eb25fcea69b8f1ca0bb056
Website	http://www.theagriverse.io / https://www.theagriverse.com/
Telegram	https://t.me/theagriverse
Twitter	https://twitter.com/TheAgriverse
Discord	https://discord.gg/paFen4Ys3s
Report Date	September 25, 2022

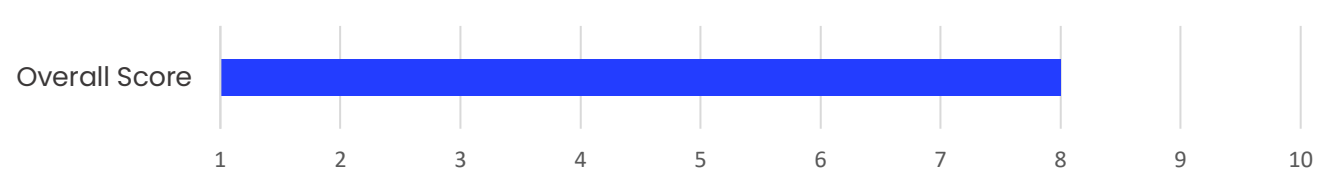
Verify the authenticity of this report on our website: https://www.interfi.network/audits

EXECUTIVE SUMMARY

InterFi has performed the automated and manual analysis of solidity codes. Solidity codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical 🔴	Major 🟠	Medium 🟡	Minor 🟢	Unknown 🟤
Open	0	0	2	3	0
Acknowledged	0	2	1	4	0
Resolved	0	0	0	0	0
Noteworthy Privileges	Blacklist, Set Buyback Limit, Set Transaction and Wallet Limit, Set Fees, Set Router Version, Set Dividend Parameters				

Agriverse's smart contract source codes have achieved the following score: **8.0**



ℹ Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

ℹ Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

TABLE OF CONTENTS

SCOPE OF WORK

InterFi was consulted by Agriverse to conduct the smart contract audit of their solidity source codes.

The audit scope of work is strictly limited to mentioned solidity file(s) only:

- Agriverse.sol

ℹ️ If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Verify the contract's deployment status below:

Public Contract Link	
https://bscscan.com/address/0x32F738738F238865f8506593c6e2613647eCE478#code	
Contract Name	Token
Compiler Version	0.8.6
License	MIT

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of InterFi's auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
  - Remix IDE Developer Tool
  - Open Zeppelin Code Analyzer
  - SWC Vulnerabilities Registry
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	○ Token Supply Manipulation ○ Access Control and Authorization ○ Assets Manipulation ○ Ownership Control ○ Liquidity Access ○ Stop and Pause Trading ○ Ownable Library Verification

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to solidity codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

ℹ️ It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical 🔴	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major 🟠	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium 🟡	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- The client can lower centralization-related risks by implementing below mentioned practices:
- Privileged role's private key must be carefully secured to avoid any potential hack.
- Privileged role should be shared by multi-signature (multi-sig) wallets.
- Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.
- Renouncing the contract ownership, and privileged roles.
- Remove functions with elevated centralization risk.

ℹ️ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💸	Function is payable
🔒	Function is internal
🔐	Function is private
!	Function is important

IERC20	Interface			
└	totalSupply	External !		NO!
└	balanceOf	External !		NO!
└	transfer	External !	🛑	NO!
└	allowance	External !		NO!
└	approve	External !	🛑	NO!
└	transferFrom	External !	🛑	NO!
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	mod	Internal 🔒		
Context	Implementation			
└	_msgSender	Internal 🔒		

└	_msgData	Internal 🔒		
SafeMathInt	Library		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	sub	Internal 🔒		
└	add	Internal 🔒		
└	abs	Internal 🔒		
└	toUint256Safe	Internal 🔒		
SafeMathUint	Library		
└	toInt256Safe	Internal 🔒		
IterableMapping	Library		
└	get	Internal 🔒		
└	getIndexOfKey	Internal 🔒		
└	getKeyAtIndex	Internal 🔒		
└	size	Internal 🔒		
└	set	Internal 🔒	🛑	
└	remove	Internal 🔒	🛑	
Address	Library		
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	_functionCallWithValue	Private 🔐	🛑	
SafeERC20	Library		
└	safeTransfer	Internal 🔒	🛑	

└	safeTransferFrom	Internal 🔒	🛑	
└	safeApprove	Internal 🔒	🛑	
└	safeIncreaseAllowance	Internal 🔒	🛑	
└	safeDecreaseAllowance	Internal 🔒	🛑	
└	_callOptionalReturn	Private 🔐	🛑	
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	geUnlockTime	Public ❗		NO❗
└	lock	Public ❗	🛑	onlyOwner
└	unlock	Public ❗	🛑	NO❗
IUniswapV2Factory	Interface			
└	feeTo	External ❗		NO❗
└	feeToSetter	External ❗		NO❗
└	getPair	External ❗		NO❗
└	allPairs	External ❗		NO❗
└	allPairsLength	External ❗		NO❗
└	createPair	External ❗	🛑	NO❗
└	setFeeTo	External ❗	🛑	NO❗
└	setFeeToSetter	External ❗	🛑	NO❗
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❗
└	getAmountOut	External ❗		NO❗
└	getAmountIn	External ❗		NO❗
└	getAmountsOut	External ❗		NO❗
└	getAmountsIn	External ❗		NO❗
IUniswapV2Router02	Interface	IUniswapV2Router01		
└	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
└	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Token	Implementation	Context, IERC20, Ownable		
└	<Constructor>	Public ❗	💵	NO❗
└	name	Public ❗		NO❗
└	updatePcsV2Router	Public ❗	🛑	onlyOwner
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	allowance	Public ❗		NO❗

└	approve	Public ❗	🛑	NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	isExcludedFromReward	Public ❗		NO❗
└	totalFees	Public ❗		NO❗
└	deliver	Public ❗	🛑	NO❗
└	reflectionFromToken	Public ❗		NO❗
└	tokenFromReflection	Public ❗		NO❗
└	excludeFromReward	Public ❗	🛑	onlyOwner
└	includeInReward	External ❗	🛑	onlyOwner
└	excludeFromFee	Public ❗	🛑	onlyOwner
└	includeInFee	Public ❗	🛑	onlyOwner
└	setAllFeePercent	External ❗	🛑	onlyOwner
└	buyBackUpperLimitAmount	Public ❗		NO❗
└	setBuybackUpperLimit	External ❗	🛑	onlyOwner
└	setMaxTxPercent	External ❗	🛑	onlyOwner
└	setMaxWalletPercent	External ❗	🛑	onlyOwner
└	setSwapAndLiquifyEnabled	Public ❗	🛑	onlyOwner
└	setFeeWallet	External ❗	🛑	onlyOwner
└	setFeeWalletCharity	External ❗	🛑	onlyOwner
└	setWalletFeeTokenType	External ❗	🛑	onlyOwner
└	setWalletCharityFeeTokenType	External ❗	🛑	onlyOwner
└	setMinimumTokenBalanceForDividends	External ❗	🛑	onlyOwner
└	<Receive Ether>	External ❗	💵	NO❗
└	_reflectFee	Private 🔐	🛑	
└	_getValues	Private 🔐		
└	_getTValues	Private 🔐		
└	_getRValues	Private 🔐		
└	_getRate	Private 🔐		
└	_getCurrentSupply	Private 🔐		

└	_takeLiquidity	Private 🔐	🛑	
└	calculateTaxFee	Private 🔐		
└	calculateLiquidityFee	Private 🔐		
└	removeAllFee	Private 🔐	🛑	
└	restoreAllFee	Private 🔐	🛑	
└	isExcludedFromFee	Public ❗		NO❗
└	_approve	Private 🔐	🛑	
└	_transfer	Private 🔐	🛑	
└	swapAndLiquify	Private 🔐	🛑	lockTheSwap
└	buyBackTokens	Private 🔐	🛑	lockTheSwap
└	swapTokensForBNB	Private 🔐	🛑	
└	swapBNBForTokens	Private 🔐	🛑	
└	swapTokensForRewardToken	Private 🔐	🛑	
└	addLiquidity	Private 🔐	🛑	
└	_tokenTransfer	Private 🔐	🛑	
└	_transferStandard	Private 🔐	🛑	
└	_transferToExcluded	Private 🔐	🛑	
└	_transferFromExcluded	Private 🔐	🛑	
└	_transferBothExcluded	Private 🔐	🛑	
└	_tokenTransferNoFee	Private 🔐	🛑	
└	transferEth	Private 🔐	🛑	
└	recoverBEP20	Public ❗	🛑	onlyOwner
└	distributeDividends	Internal 🔒	🛑	
└	withdrawDividend	Public ❗	🛑	NO❗
└	_withdrawDividendOfUser	Internal 🔒	🛑	
└	dividendOf	Public ❗		NO❗
└	withdrawableDividendOf	Public ❗		NO❗
└	withdrawnDividendOf	Public ❗		NO❗
└	accumulativeDividendOf	Public ❗		NO❗
└	_dtransfer	Internal 🔒	🛑	
└	_dmint	Internal 🔒	🛑	

| └ | _dburn | Internal 🔒 | 🛑 | | |

| └ | _setBalance | Internal 🔒 | 🛑 | | |

| └ | excludeFromDividends | Public ❗ | 🛑 | onlyOwner |

| └ | updateClaimWait | External ❗ | 🛑 | onlyOwner |

| └ | getLastProcessedIndex | External ❗ | | NO❗ |

| └ | getNumberOfDividendTokenHolders | External ❗ | | NO❗ |

| └ | getAccountDividendsInfo | Public ❗ | | NO❗ |

| └ | getAccountDividendsInfoAtIndex | Public ❗ | | NO❗ |

| └ | canAutoClaim | Private 🔐 | | |

| └ | setBalance | Private 🔐 | 🛑 | | |

| └ | process | Public ❗ | 🛑 | NO❗ |

| └ | processAccount | Internal 🔒 | 🛑 | | |

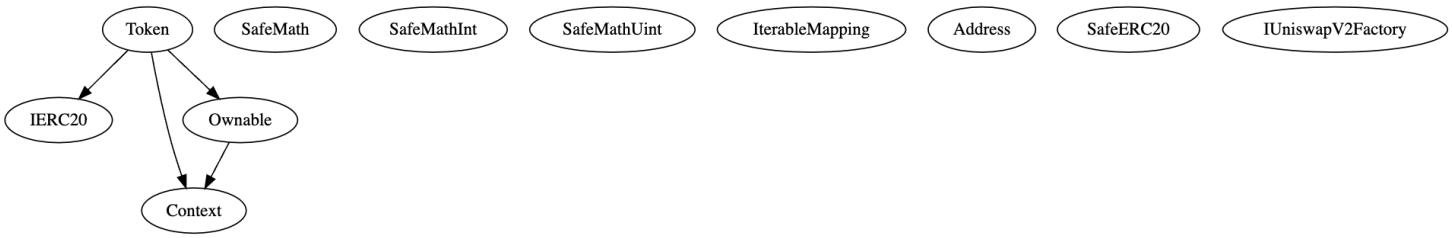
| └ | updateGasForProcessing | Public ❗ | 🛑 | onlyOwner |

| └ | processDividendTracker | External ❗ | 🛑 | NO❗ |

| └ | blacklistAddress | External ❗ | 🛑 | onlyOwner |

| └ | claim | External ❗ | 🛑 | NO❗ |

INHERITANCE GRAPH



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralization privileges of Agriverse	Major 🟠

Centralized privileges are listed below:

```
renounceOwnership
transferOwnership
lock
updatePcsV2Router
excludeFromReward
excludeFromFee
setAllFeePercent
setBuybackUpperLimit
setMaxTxPercent
setMaxWalletPercent
setSwapAndLiquifyEnabled
setFeeWallet
setFeeWalletCharity
setWalletFeeTokenType
setWalletCharityFeeTokenType
setMinimumTokenBalanceForDividends
recoverBEP20
excludeFromDividends
updateClaimWait
updateGasForProcessing
blacklistAddress
```


RECOMMENDATION

Deployer and/or contract owner private keys are secured carefully. Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding.

Identifier	Definition	Severity
CEN-02	Initial asset distribution	Medium 🟡

All of the initially minted assets are sent to the contract deployer when deploying the contract. This can be an issue as the deployer and/or contract owner can distribute tokens without consulting the community.

RECOMMENDATION

Deployer and/or contract owner private keys are secured carefully. Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding. The community should be consulted during the initial asset distribution process.

Identifier	Definition	Severity
CEN-03	Privileged role performing blacklist	Major 🟠

Privileged role can call `blacklist()`

```
function blacklistAddress(address account, bool value) external onlyOwner {
    _isBlacklisted[account] = value;
}
```


RECOMMENDATION

Remove blacklist – as it can intentionally stop an address from accessing smart contract function modules.

Identifier	Definition	Severity
LOG-01	Lack of arbitrary limits	Minor

Below mentioned functions are set without any arbitrary limits.

`setBuybackUpperLimit()`

Below mentioned functions are set with a high arbitrary limit.

`setAllFeePercent()`

RECOMMENDATION

These functions should be provided arbitrary limits, e.g., put a `require` check that allows maximum tax change up to 25%.

Identifier	Definition	Severity
LOG-02	Potential sandwich attack	Minor

Potential sandwich attack happens when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by front-running a transaction to purchase assets and make profits by back-running a transaction to sell assets. Below mentioned functions are called without setting restrictions on slippage or minimum output:

```
addLiquidity()
swapTokensForRewardToken()
swapBNBForTokens()
swapTokensForBNB()
```


RECOMMENDATION

These functions should be provided reasonable minimum output amounts, instead of zero. Read more: https://coinmarketcap.com/alexandria/article/what-are-sandwich-attacks-in-defi-and-how-can-you-avoid-them

Identifier	Definition	Severity
LOG-04	Possible to regain ownership after renouncing contract ownership	Medium 🟡

Smart contract owner can gain ownership even after `renounceOwnership()` is called. `lock`, and `unlock` functions can be used to set `_previousOwner` as current owner, and then renouncing the ownership.

```
    //Locks the contract for owner for the amount of time provided
    function lock(uint256 time) public virtual onlyOwner {
        _previousOwner = _owner;
        _owner = address(0);
        _lockTime = block.timestamp + time;
        emit OwnershipTransferred(_owner, address(0));
    }

    //Unlocks the contract for owner when _lockTime is exceeds
    function unlock() public virtual {
        require(_previousOwner == msg.sender, "You don't have permission to unlock the token
contract");
        require(block.timestamp > _lockTime , "Contract is locked until 7 days");
        emit OwnershipTransferred(_owner, _previousOwner);
        _owner = _previousOwner;
    }
```

RECOMMENDATION

Remove lock contract, and unlock contract options. This logical error was identified in BSC#0xc748673057861a797275CD8A068AbB95A902e8de.

Identifier	Definition	Severity
COD-01	Authorization through `tx.origin`	Medium 🟡

Using `tx.origin` for authorization could make the contract vulnerable as it refers to the original external account that started the transaction.

RECOMMENDATION

Avoid authorizations via global variables wherever necessary.

Identifier	Definition	Severity
COD-02	Timestamp manipulation via `block.timestamp` Avoid using `block.number` as timestamp	Minor 🟢

Be aware that the timestamp of the block can be manipulated by a miner. When the contract uses the timestamp to seed a random number, the miner can actually post a timestamp within 15 seconds of the block being validated, effectively allowing the miner to precompute an option more favorable to their chances, this is a critical exploit for contracts calculating random numbers, e.g., lottery.

RECOMMENDATION

To maintain block integrity, follow 15 seconds rule, and scale time dependent events accordingly.

Identifier	Definition	Severity
COD-04	Unclear error messages	

RECOMMENDATION

Provide accurate information strings for `require` related errors.

Identifier	Definition	Severity
COD-10	Third Party Dependencies	Minor

Smart contract is interacting with third party protocols e.g., Pancakeswap, Uniswap. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary.

Identifier	Definition	Severity
COD-11	Public functions that can be declared external	Minor

Smart contract Agriverse.sol may have access control vulnerabilities.

RECOMMENDATION

Public functions that are never called by the contract should be declared external to optimize gas usage.

Identifier	Definition	Severity
VOL-01	Irrelevant code	Minor

Redundant code lines are found in contract, some instances are highlighted below:

```
SafeMathInt.div
Address.sendValue
Address.functionCallWithValue
SafeMathInt.abs
SafeERC20._callOptionalReturn
SafeERC20.safeTransfer
SafeMathInt.mul
SafeERC20.safeIncreaseAllowance
IterableMapping.get
SafeERC20.safeApprove
```


RECOMMENDATION

Remove redundant and dead code.

Identifier	Definition	Severity
COM-02	Outdated compiler version	

Compiler is set an outdated version.

RECOMMENDATION

Set Compiler to version 0.8.12 or above.

Identifier	Definition	Severity
COM-03	Hardcoded gas amount	Minor 🟢

Gas amount is set to `gasForProcessing = 300000;`

```
    function processDividendTracker(uint256 gas) external {
        (uint256 iterations, uint256 claims, uint256 _lastProcessedIndex) = process(gas);
        emit ProcessedDividendTracker(iterations, claims, _lastProcessedIndex, false, gas,
tx.origin);
    }
```


RECOMMENDATION

Stop the `dividendTracker.process()` call in the `processDividendTracker().` Users should claim their rewards manually through the function `claim().`

DISCLAIMERS

InterFi Network provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. 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. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way

to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, INTERFI NETWORK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. InterFi Network does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than InterFi Network. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' and social accounts' owners. You agree that InterFi Network is not responsible for the content or operation of such websites and social accounts and that InterFi Network shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT INTERFI NETWORK

InterFi Network provides intelligent blockchain solutions. We provide solidity development, testing, and auditing services. We have developed 150+ solidity codes, audited 1000+ smart contracts, and analyzed 500,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Velas, Oasis, etc.

InterFi Network is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 6+ casual contributors.

Website: https://interfi.network

Email: hello@interfi.network

GitHub: https://github.com/interfinetwork

Telegram (Engineering): https://t.me/interfiaudits

Telegram (Onboarding): https://t.me/interfisupport

interfinetwork

hello@interfi.network

https://interfi.network

SMART CONTRACT AUDITS | SOLIDITY DEVELOPMENT AND TESTING

RELENTLESSLY SECURING PUBLIC AND PRIVATE BLOCKCHAINS