SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Stag
$STAG
BEP 20

0xa94D583e4Ea69216b870A6300a9f717bB6D4a076

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	Stag
Ticker/Simbol	$STAG
Blockchain	Binance Smart Chain BEP20
Contract Address	0xa94D583e4Ea69216b870A6300a9f717bB6D4a076
Creator Address	0x4E4506e74aCa9be58C3D6CE03098f6c329Dd7158
Current Owner Address	0x85e7C32DC8bb5b2F74c17a1e0F64De3b940D1289
Contract Explorer	https://bscscan.com/address/0xa94d583e4ea69216b870a6300a9f717bb6d4a076#code
Compiler Version	v0.8.19+commit.7dd6d404
License	MIT
Optimisation	Yes with 200 Runs
Total Supply	100,000 **$STAG**
Decimals	18

Creation/Audit

Contract Deployed	06 Oct 2023
Audit Created	26 Nov 2023
Audit Update	V 1.0

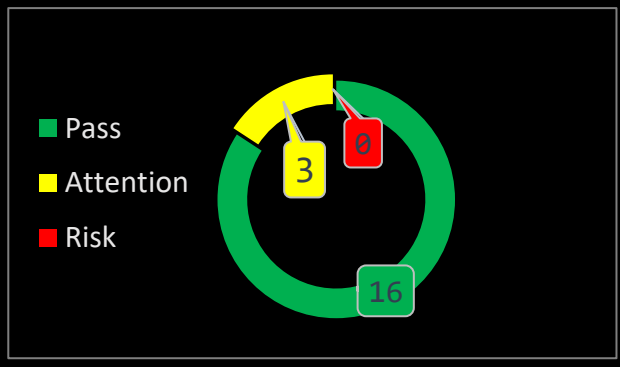
Verified Socials

Website	https://stagtoken.com/
Telegram	https://t.me/StagToken
Twitter (X)	https://x.com/STAG_BSC

STAG BEP20

Contract Function Analysis

Pass Attention Item Risky Item



Contract Verified	✔	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0x85e7C32DC8bb5b2F74c17a1e0F64De3b940D1289
Buy Tax	8 %	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	10 %	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	✔	LP Lock Status on 26.11.2023: 99.72% Mudra Locker for 327 days
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	⚠	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	✅	No Blacklist Setting function found. 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	⚠️	Max Transaction and Holding Modify function found. 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

⚠ Set Fee

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

```
ftrace | funcSig
function updateBuyTax(uint48 _marketingTax↑, uint48 _liquidityTax↑, uint48 _rewardTax↑) external onlyOwner {
    Taxes memory taxes;
    taxes.marketingTax = _marketingTax↑;
    taxes.liquidityTax = _liquidityTax↑;
    taxes.rewardTax = _rewardTax↑;
    taxes.totalTax = _marketingTax↑ + _liquidityTax↑ + _rewardTax↑;

    emit UpdatedBuyTax(taxes.totalTax);
    buyTax = taxes;
}

ftrace | funcSig
function updateSellTax(uint48 _marketingTax↑, uint48 _liquidityTax↑, uint48 _rewardTax↑) external onlyOwner {
    Taxes memory taxes;
    taxes.marketingTax = _marketingTax↑;
    taxes.liquidityTax = _liquidityTax↑;
    taxes.rewardTax = _rewardTax↑;
    taxes.totalTax = _marketingTax↑ + _liquidityTax↑ + _rewardTax↑;

    emit UpdatedSellTax(taxes.totalTax);
    sellTax = taxes;
}
```

⚠ Whitelist Function

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
function setExemptFromFee(address _address↑, bool _isExempt↑) external onlyOwner {
    require(_address↑ != address(0), "Zero Address");
    require(_address↑ != address(this), "Cannot unexempt contract");
    exemptFromFees[_address↑] = _isExempt↑;
    emit SetExemptFromFees(_address↑, _isExempt↑);
}

ftrace | funcSig
function setExemptFromLimit(address _address↑, bool _isExempt↑) external onlyOwner {
    require(_address↑ != address(0), "Zero Address");
    if(!_isExempt↑){
        require(_address↑ != lpPair, "Cannot remove pair");
    }
    exemptFromLimits[_address↑] = _isExempt↑;
    emit SetExemptFromLimits(_address↑, _isExempt↑);
}
```

⚠ Max Transaction and Holding Modify Function

Max Transaction and Holding Modify function found.

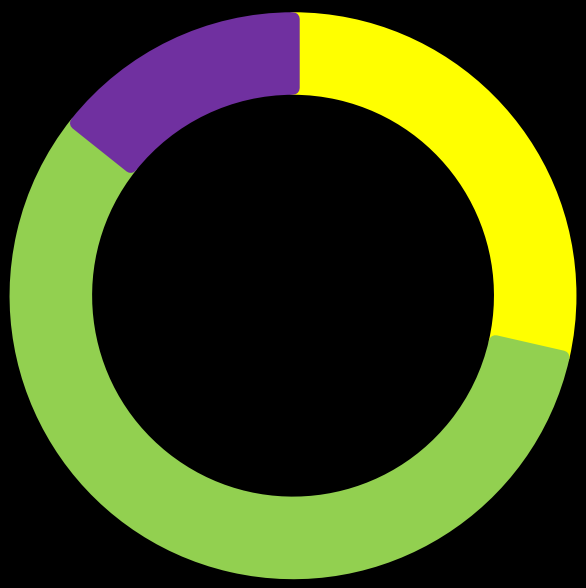
If there is a function for this, the maximum trading amount or maximum

```
ftrace | funcSig
function updateTransactionLimit(uint128 newNumInTokens) external onlyOwner {

    txLimits.transactionLimit = uint128(newNumInTokens * (10**decimals()));
    emit UpdatedTransactionLimit(txLimits.transactionLimit);
}

ftrace | funcSig
function updateWalletLimit(uint128 newNumInTokens) external onlyOwner {

    txLimits.walletLimit = uint128(newNumInTokens * (10**decimals()));
    emit UpdatedWalletLimit(txLimits.walletLimit);
}
```

Contract Security

Total Findings: 7

- **High 0**
- **Medium 2**
- **Low 4**
- **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: (2)

- Incorrect Acces Control
- Approve of Front Running Attack

Low severity issues: (4)

- Missing Events
- Long Number Literals
- Floating Pragma
- Return value of low level calls

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 SPECIFIC TO SMART CONTRACTS.

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	Low	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	Passed	Passed	Passed
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	**Passed**	**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.

⚠ Incorrect Acces Control (3 Item)

Item: 1	Location:	Line 814-825	Severity:	Medium

Function	Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract DividendPayingContract is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function distributeDividends is missing the modifier onlyOwner.
Remedation	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
814
        ftrace | funcSig
815     function distributeDividends() public override payable {
816         if(totalBalance > 0 && msg.value > 0){
817             magnifiedDividendPerShare = magnifiedDividendPerShare.add(
818                 (msg.value).mul(magnitude) / totalBalance
819             );
820             emit DividendsDistributed(msg.sender, msg.value);
821
822             totalDividendsDistributed = totalDividendsDistributed.add(msg.value);
823         }
824     }
825
```

Item: 2	Location:	Line 1316-1320	Severity:	Medium

Function	The contract NetronProtocol is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function claim is missing the modifier onlyOwner.
Remedation	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
1316
ftrace | funcSig
1317 function claim() external {
1318     dividendTracker.processAccount(payable(msg.sender), false);
1319 }
1320
```

Item: 3	Location:	Line 1353-1362	Severity:	Medium

Function	The contract NetronProtocol is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function withdrawDividendToken is missing the modifier onlyOwner.
Remedation	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
1353
        ftrace | funcSig
1354    function withdrawDividendToken(uint256 minAmount) external {
1355        uint256 amountEthForrewards = dividendTracker.withdrawDividendOfUserFor(payable(msg.sender));
1356        if(amountEthForrewards > 0){
1357            buyBackTokens(amountEthForrewards, minAmount, msg.sender);
1358        } else {
1359            revert("No rewards");
1360        }
1361    }
1362
```

⚠ Aprrove of Front Running Attack (2 Item)

Item: 1	Location:	Line 208-214	Severity:	Medium

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

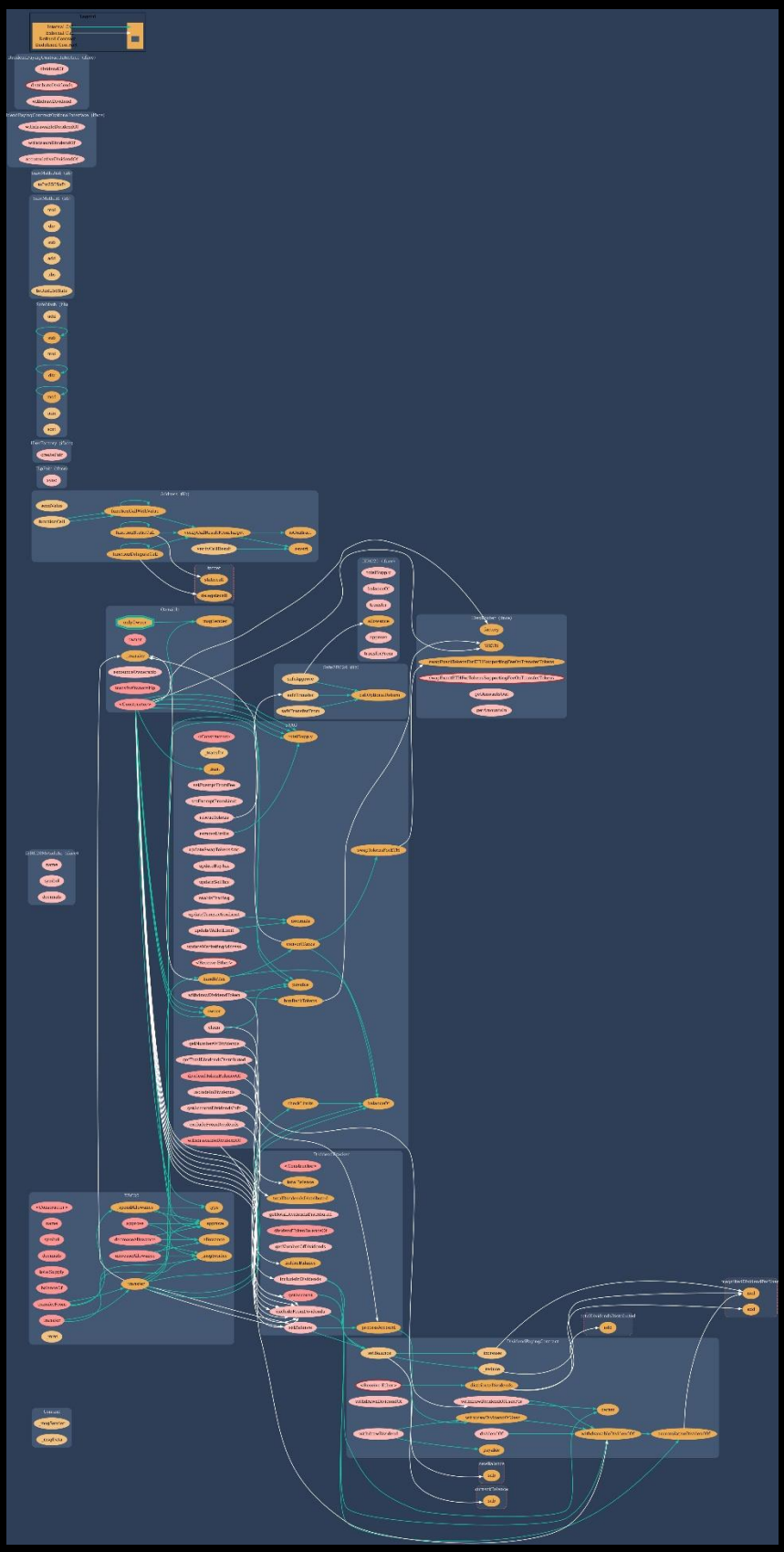
```
ftrace | funcSig
209 function approve(address spender, uint256 amount) public virtual override returns (bool) {
210     address owner = _msgSender();
211     _approve(owner, spender, amount);
212     return true;
213 }
214
```

Item: 2	Location:	Line 360-368	Severity:	Medium

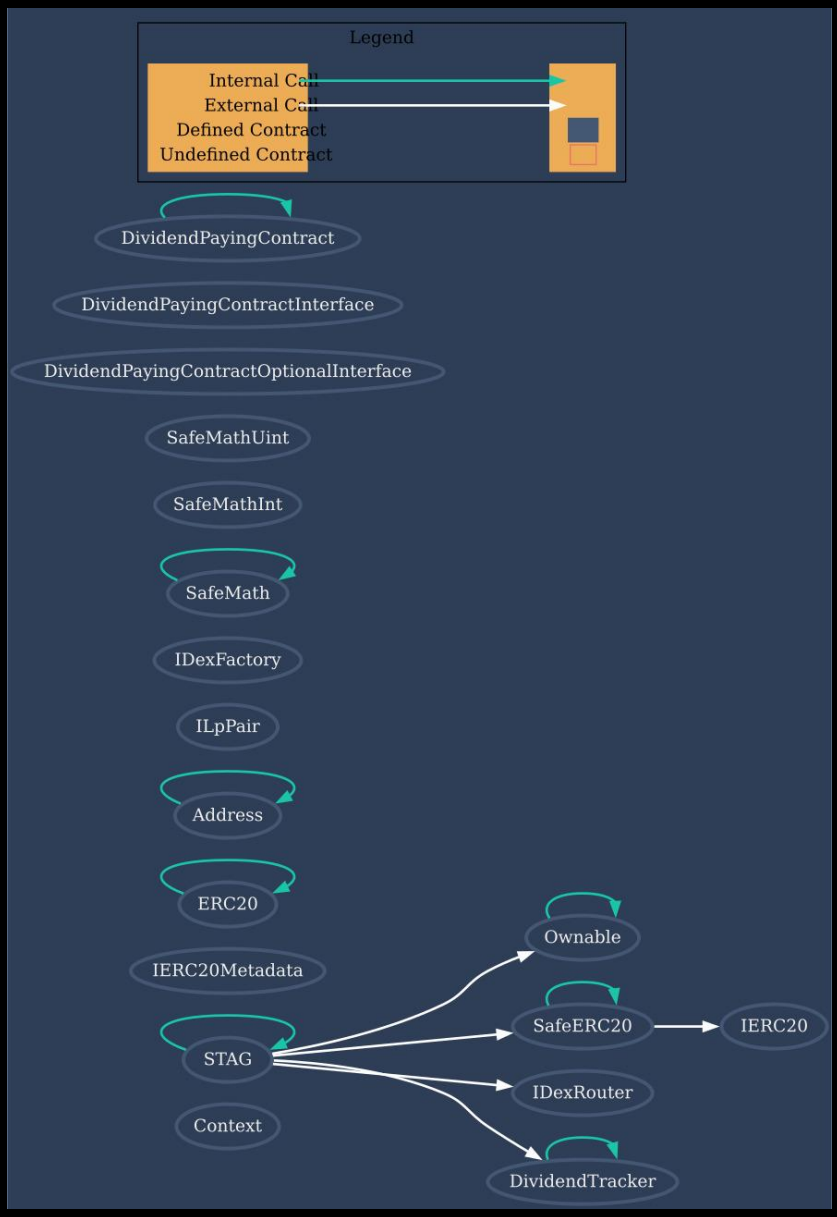
Function	The _spendAllowance() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver 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 _spendAllowance 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

```
360     function _spendAllowance(address owner, address spender, uint256 amount) internal virtual {
361         uint256 currentAllowance = allowance(owner, spender);
362         if (currentAllowance != type(uint256).max) {
363             require(currentAllowance >= amount, "ERC20: insufficient allowance");
364             unchecked {
365                 _approve(owner, spender, currentAllowance - amount);
366             }
367         }
368     }
369 }
370
```

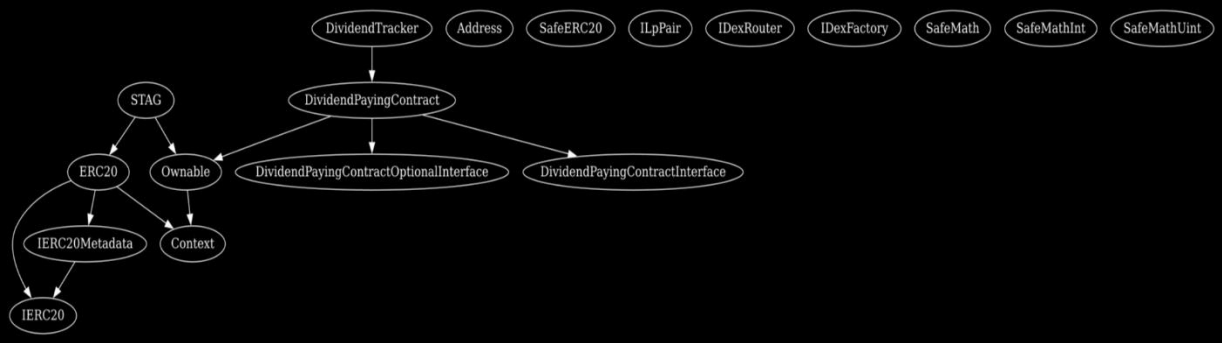
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
L	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
IERC20	Interface			
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!
L	increaseAllowance	Public !		NO!
L	decreaseAllowance	Public !		NO!
L	_transfer	Internal		
L	_mint	Internal		
L	_approve	Internal		
L	_spendAllowance	Internal		
Ownable	Implementation	Context		
L		Public !		NO!
L	owner	Public !		NO!
L	renounceOwnership	External !		onlyOwner
L	transferOwnership	Public !		onlyOwner
Address	Library			

L	isContract	Internal		
L	sendValue	Internal		
L	functionCall	Internal		
L	functionCall	Internal		
L	functionCallWithValue	Internal		
L	functionCallWithValue	Internal		
L	functionStaticCall	Internal		
L	functionStaticCall	Internal		
L	functionDelegateCall	Internal		
L	functionDelegateCall	Internal		
L	verifyCallResultFromTarget	Internal		
L	verifyCallResult	Internal		
L	_revert	Private		
SafeERC20	Library			
L	safeTransfer	Internal		
L	safeTransferFrom	Internal		
L	_callOptionalReturn	Private		
L	safeApprove	Internal		

ILpPair	Interface			
L	sync	External !		NO!
IDexRouter	Interface			
L	factory	External !		NO!
L	WETH	External !		NO!
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External !		NO!
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External !		NO!
L	getAmountsOut	External !		NO!
L	getAmountsIn	External !		NO!
IDexFactory	Interface			
L	createPair	External !		NO!
SafeMath	Library			
L	add	Internal		
L	sub	Internal		
L	sub	Internal		
L	mul	Internal		
L	div	Internal		
L	div	Internal		
L	mod	Internal		

L	mod	Internal 🔒		
L	min	Internal 🔒		
L	sqrt	Internal 🔒		
SafeMathInt	Library			
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	sub	Internal 🔒		
L	add	Internal 🔒		
L	abs	Internal 🔒		
L	toUint256Safe	Internal 🔒		
SafeMathUint	Library			
L	toInt256Safe	Internal 🔒		
DividendPayingContractOptionalInterface	Interface			
L	withdrawableDividendOf	External ❗		NO❗
L	withdrawnDividendOf	External ❗		NO❗
L	accumulativeDividendOf	External ❗		NO❗
DividendPayingContractInterface	Interface			
L	dividendOf	External ❗		NO❗

L	distributeDividends	External		NO
L	withdrawDividend	External		NO
DividendPayingContract	Implementation	DividendPayingContractInterface, DividendPayingContractOptionalInterface, Ownable		
L		External		NO
L	distributeDividends	Public		NO
L	withdrawDividend	External		NO
L	_withdrawDividendOfUser	Internal		
L	withdrawDividendOfUserFor	External		onlyOwner
L	dividendOf	External		NO
L	withdrawableDividendOf	Public		NO
L	withdrawnDividendOf	External		NO
L	accumulativeDividendOf	Public		NO
L	_increase	Internal		
L	_reduce	Internal		
L	_setBalance	Internal		

DividendTracker	Implementation	DividendPaying Contract		
L		Public !		NO!
L	getAccount	Public !		NO!
L	setBalance	External !		onlyOwner
L	processAccount	Public !		onlyOwner
L	getTotalDividendsDistributed	External !		NO!
L	dividendTokenBalanceOf	Public !		NO!
L	getNumberOfDividends	External !		NO!
L	excludeFromDividends	External !		onlyOwner
L	includeInDividends	External !		onlyOwner
STAG	Implementation	ERC20, Ownable		
L		Public !		ERC20
L	_transfer	Internal		
L	checkLimits	Internal		
L	handleTax	Internal		
L	swapTokensForETH	Private		
L	convertTaxes	Private		
L	setExemptFromFee	External !		onlyOwner
L	setExemptFromLimit	External !		onlyOwner

L	updateTransactionLimit	External !		onlyOwner
L	updateWalletLimit	External !		onlyOwner
L	updateSwapTokensAmt	External !		onlyOwner
L	updateBuyTax	External !		onlyOwner
L	updateSellTax	External !		onlyOwner
L	enableTrading	External !		onlyOwner
L	removeLimits	External !		onlyOwner
L	rescueTokens	External !		onlyOwner
L	updateMarketingAddress	External !		onlyOwner
L		External !		NO!
L	claim	External !		NO!
L	getTotalDividendsDistributed	External !		NO!
L	withdrawableDividendOf	Public !		NO!
L	dividendTokenBalanceOf	Public !		NO!
L	getAccountDividendsInfo	External !		NO!
L	getNumberOfDividends	External !		NO!
L	excludeFromDividends	External !		onlyOwner
L	includeInDividends	External !		onlyOwner

L	withdrawDividendToken	External		NO!
L	buyBackTokens	Internal		

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