SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

BNB ETF TOKEN

$BNTF

BEP20

0xCC69c4488C1671c88c75eD20e465289de9eD8

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	BNB_ETF_TOKEN
Ticker/Simbol	BNTF
Blockchain	Binance Smart Chain BEP20
Contract Address	0xCC69c4488C1671c88c75eD20e465289de9eD8E05
Creator Address	0x3dc8BF63e5b5c1f0e536C27d85a06CF1f0e53D6f
Current Owner Address	0x3dc8BF63e5b5c1f0e536C27d85a06CF1f0e53D6f
Contract Explorer	https://bscscan.com/address/0xCC69c4488C1671c88c75eD20e465289de9eD8E05#code
Compiler Version	v0.8.19+commit.7dd6d404
License	MIT
Optimisation	Yes with 200 Runs
Total Supply	79,807,242.014502 **BNTF**
Decimals	18

Creation/Audit

Contract Deployed	28.05.2024
Audit Created	29.05.2024
Audit Update	V 1.0

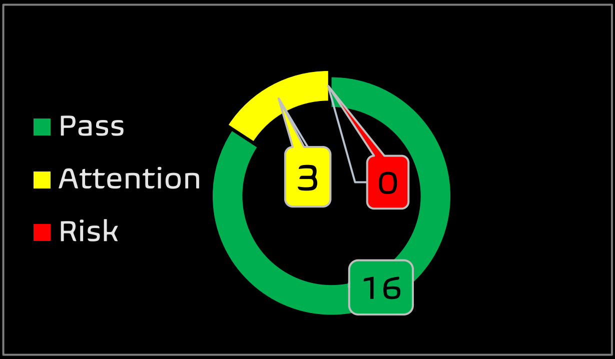
Verified Socials

Website	https://www.bnbetftoken.com/
Telegram	https://t.me/BNBETFTOKEN
Twitter (X)	https://x.com/bnbetftoken

BNB ETF TOKEN BEP20

Contract Function Analysis

✓ Pass ⚠ Attention Item ❗ Risky Item



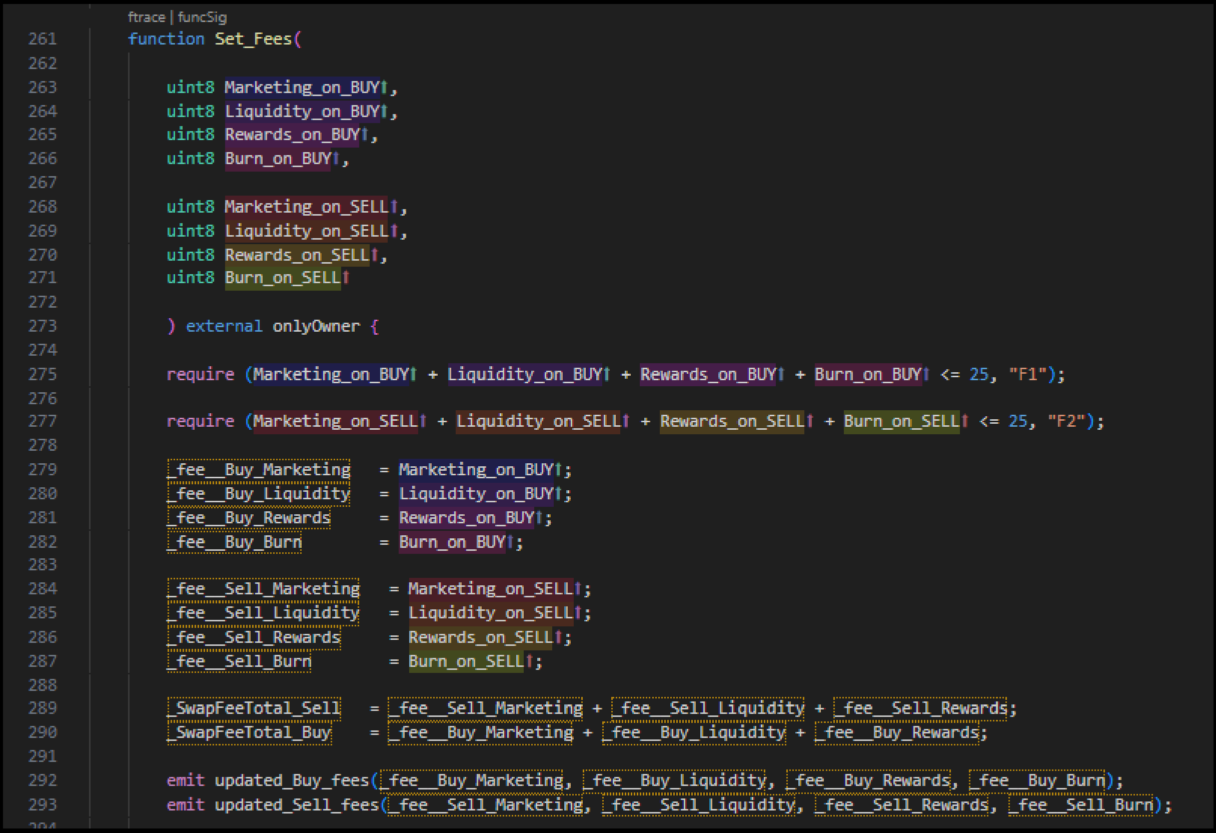
Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0x3dc8BF63e5b5c1f0e536C27d85a06CF1f0e53D6f Deployer
Buy Tax	10 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More 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	✓	Liqudity status on 29.05.2024 99% of initial liqudity locked for 92 Days on PinkSale Locker https://bscscan.com/tx/0x1776d8cb9fc8a499e7401e32cdf6f516425ceda3a71861d2b1e53f86ecc2f9c0
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	⚠ max 25%	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

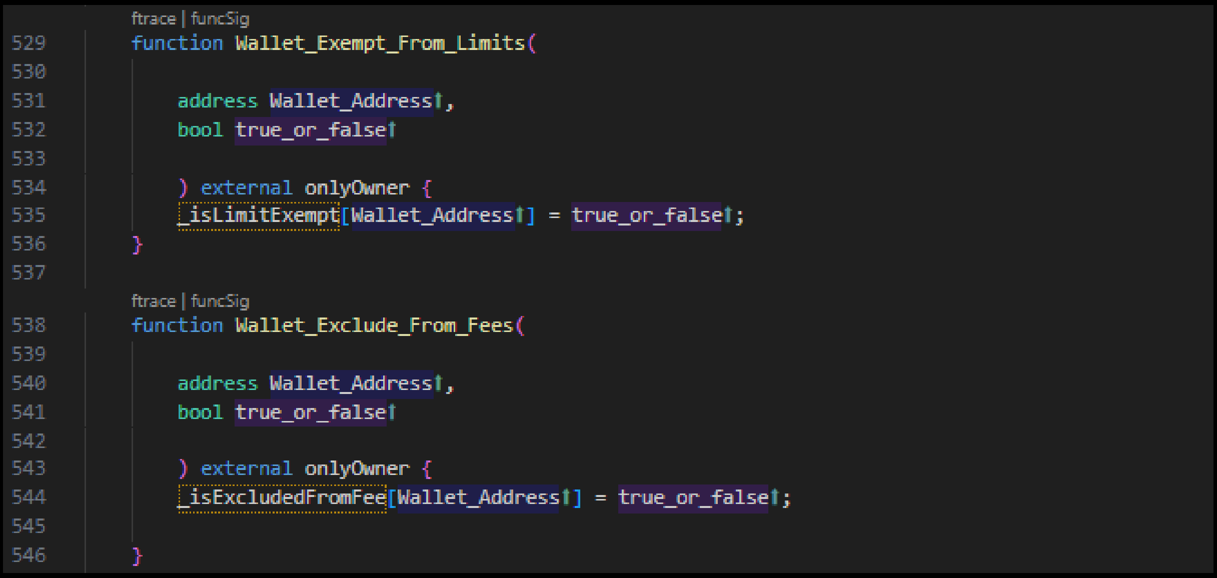
The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk).



```
ftrace | funcSig
function Set_Fees(

    uint8 Marketing_on_BUY,
    uint8 Liquidity_on_BUY,
    uint8 Rewards_on_BUY,
    uint8 Burn_on_BUY,

    uint8 Marketing_on_SELL,
    uint8 Liquidity_on_SELL,
    uint8 Rewards_on_SELL,
    uint8 Burn_on_SELL

    ) external onlyOwner {

    require (Marketing_on_BUY + Liquidity_on_BUY + Rewards_on_BUY + Burn_on_BUY <= 25, "F1");

    require (Marketing_on_SELL + Liquidity_on_SELL + Rewards_on_SELL + Burn_on_SELL <= 25, "F2");

    _fee__Buy_Marketing    = Marketing_on_BUY;
    _fee__Buy_Liquidity    = Liquidity_on_BUY;
    _fee__Buy_Rewards      = Rewards_on_BUY;
    _fee__Buy_Burn         = Burn_on_BUY;

    _fee__Sell_Marketing    = Marketing_on_SELL;
    _fee__Sell_Liquidity    = Liquidity_on_SELL;
    _fee__Sell_Rewards      = Rewards_on_SELL;
    _fee__Sell_Burn         = Burn_on_SELL;

    _SwapFeeTotal_Sell   = _fee__Sell_Marketing + _fee__Sell_Liquidity + _fee__Sell_Rewards;
    _SwapFeeTotal_Buy    = _fee__Buy_Marketing + _fee__Buy_Liquidity + _fee__Buy_Rewards;

    emit updated_Buy_fees(_fee__Buy_Marketing, _fee__Buy_Liquidity, _fee__Buy_Rewards, _fee__Buy_Burn);
    emit updated_Sell_fees(_fee__Sell_Marketing, _fee__Sell_Liquidity, _fee__Sell_Rewards, _fee__Sell_Burn);
```

⚠ Whitelist

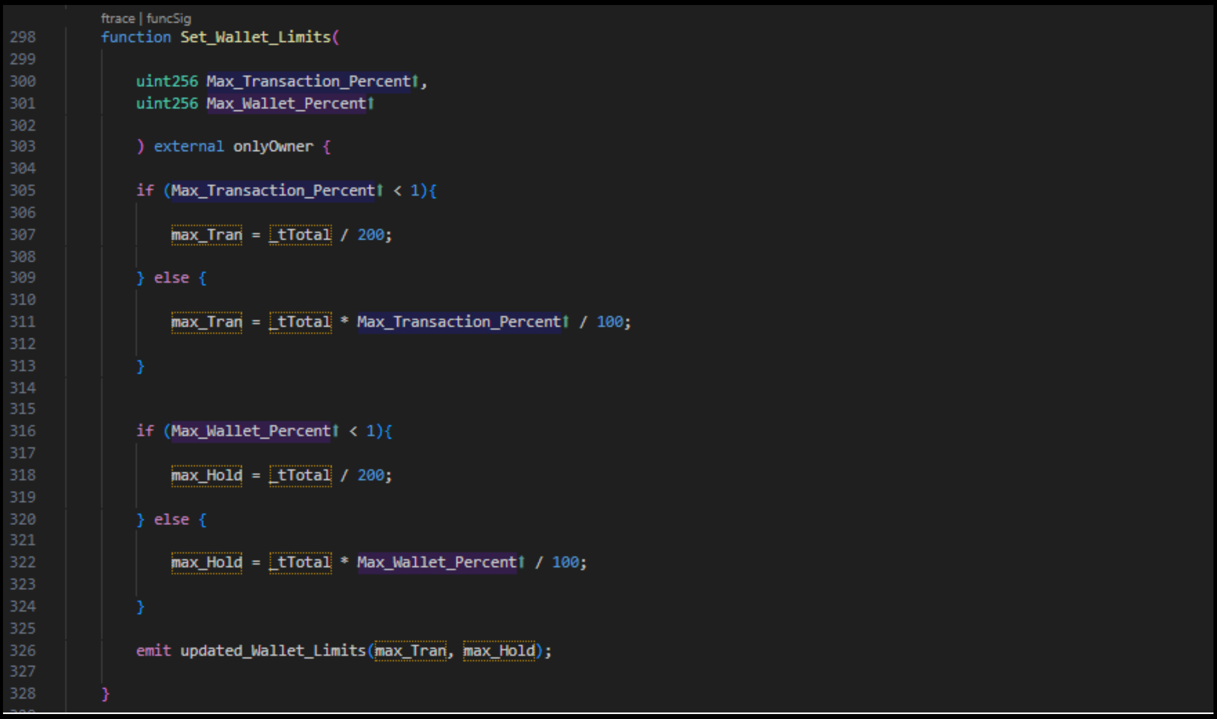
If there is a function for this Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee. Can cause farming)



```
ftrace | funcSig
function Wallet_Exempt_From_Limits(

    address Wallet_Address,
    bool true_or_false

    ) external onlyOwner {
    _isLimitExempt[Wallet_Address] = true_or_false;
}

ftrace | funcSig
function Wallet_Exclude_From_Fees(

    address Wallet_Address,
    bool true_or_false

    ) external onlyOwner {
    _isExcludedFromFee[Wallet_Address] = true_or_false;

}
```

⚠ Max Transaction and Holding Modify function

If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot



```
ftrace | funcSig
function Set_Wallet_Limits(

    uint256 Max_Transaction_Percent,
    uint256 Max_Wallet_Percent

    ) external onlyOwner {

    if (Max_Transaction_Percent < 1){

        max_Tran = _tTotal / 200;

    } else {

        max_Tran = _tTotal * Max_Transaction_Percent / 100;

    }


    if (Max_Wallet_Percent < 1){

        max_Hold = _tTotal / 200;

    } else {

        max_Hold = _tTotal * Max_Wallet_Percent / 100;

    }

    emit updated_Wallet_Limits(max_Tran, max_Hold);

}
```

Contract Security

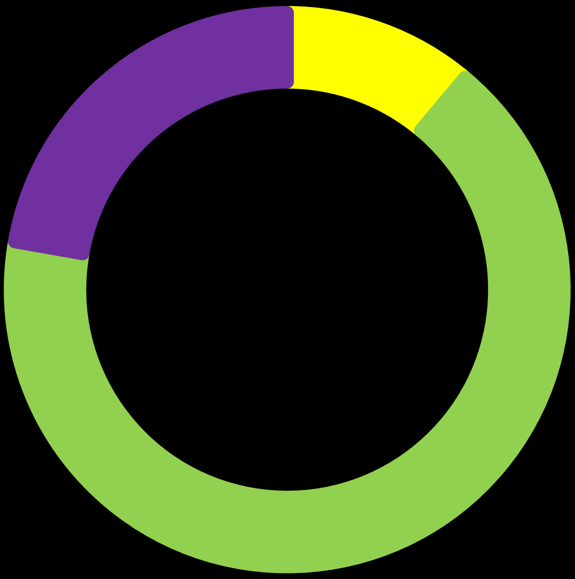
Total Findings: 9

🟥 High 0

🟨 Medium 1

🟩 Low 6

🟪 Info 2



🟥 **High Severity Issues:** High possibility to cause problems, need to be resolved.

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

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

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

Contract Security

List of Found Issues

🟥 High severity Issues: (0)

🟨 Medium severity issues: (1)

- Reentrancy

🟩 Low severity issues: (6)

- Missing Events
- Low level calls
- Long number literals
- Outdated compiler Version
- Unchecked Array Lenght
- Approve of front running attack (Also known as Sendwich Bots attack)

🟪 Informational severity issues: (2)

- Public Functions Should be Declared External
- State Variables Should be Declared Constant

Contract Weakness Classisication

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

ID	Description	AI	Manual	Result
SWC-100	Function Default Visibility	Passed	Passed	Passed
SWC-101	Integer Overflow and Underflow	Passed	Passed	Passed
SWC-102	Outdated Compiler Version	low	low	low
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	High	Medium	Medium
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

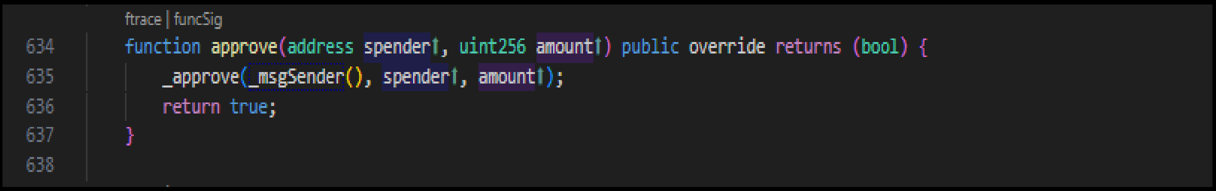
SWC-121	Missing Protection against Signature Replay Attacks	Passed	Passed	Passed
SWC-122	Lack of Proper Signature Verification	Passed	Passed	Passed
SWC-123	Requirement Violation	Passed	Passed	Passed
SWC-124	Write to Arbitrary Storage Location	Passed	Passed	Passed
SWC-125	Incorrect Inheritance Order	Passed	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	low	Passed	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed	Passed	Passed
SWC-131	Presence of unused variables	Passed	Passed	Passed
SWC-132	Unexpected Ether balance	Passed	Passed	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed	Passed	Passed
SWC-134	Message call with hardcoded gas amount	Passed	Passed	Passed
SWC-135	Code With No Effects	Passed	Passed	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed	Passed	Passed

Detected High and Medium Severity Vulnerability Description.

⚠ Approve of front running attack. Also known as Sandwich bot attack. (2 Items)

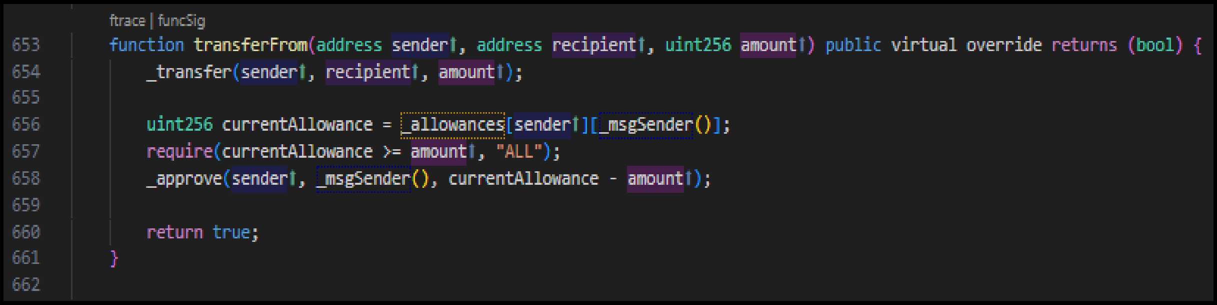
Item: 1	Location:	Line 634-637	Severity:	🟩 Low

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


```
ftrace | funcSig
634  function approve(address spender, uint256 amount) public override returns (bool) {
635      _approve(_msgSender(), spender, amount);
636      return true;
637  }
638
```

Item: 2	Location:	Line 653-661	Severity:	🟩 Low

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

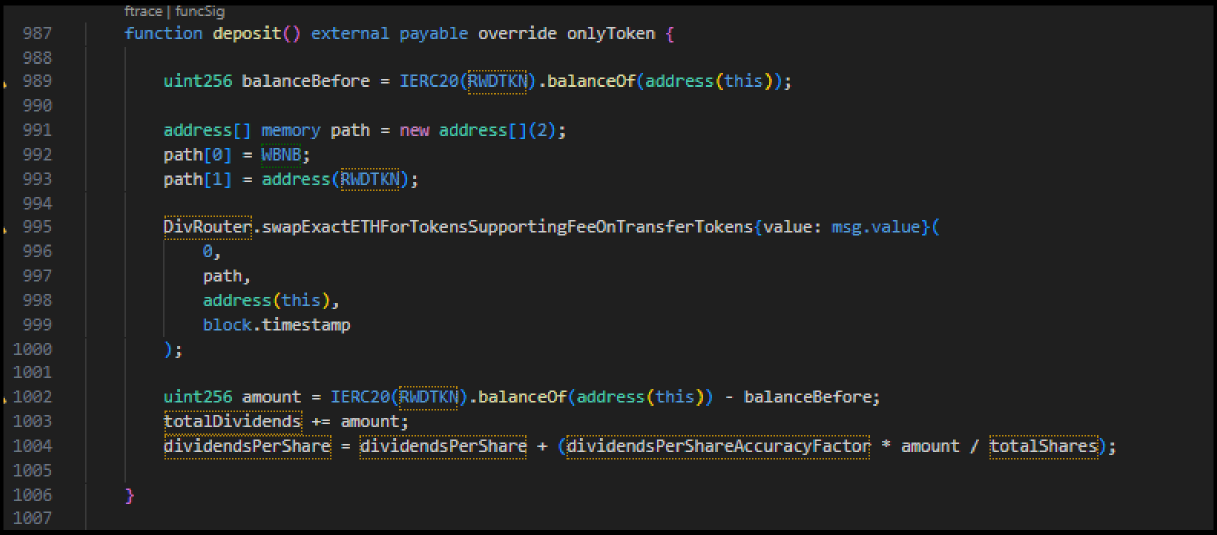


```
    ftrace | funcSig
653 function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
654     _transfer(sender, recipient, amount);
655
656     uint256 currentAllowance = _allowances[sender][_msgSender()];
657     require(currentAllowance >= amount, "ALL");
658     _approve(sender, _msgSender(), currentAllowance - amount);
659
660     return true;
661 }
662
```

⚠ Reentrancy (1 Item)

Item: 1	Location:	Line 987-1006	Severity:	🟨 Medium

Function	In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls. This may lead to loss of funds, improper value updates, token loss, etc.
Remedation	• Ensure all state changes happen before calling external contracts, i.e., update balances or code internally before calling external code • Use function modifiers that prevent reentrancy



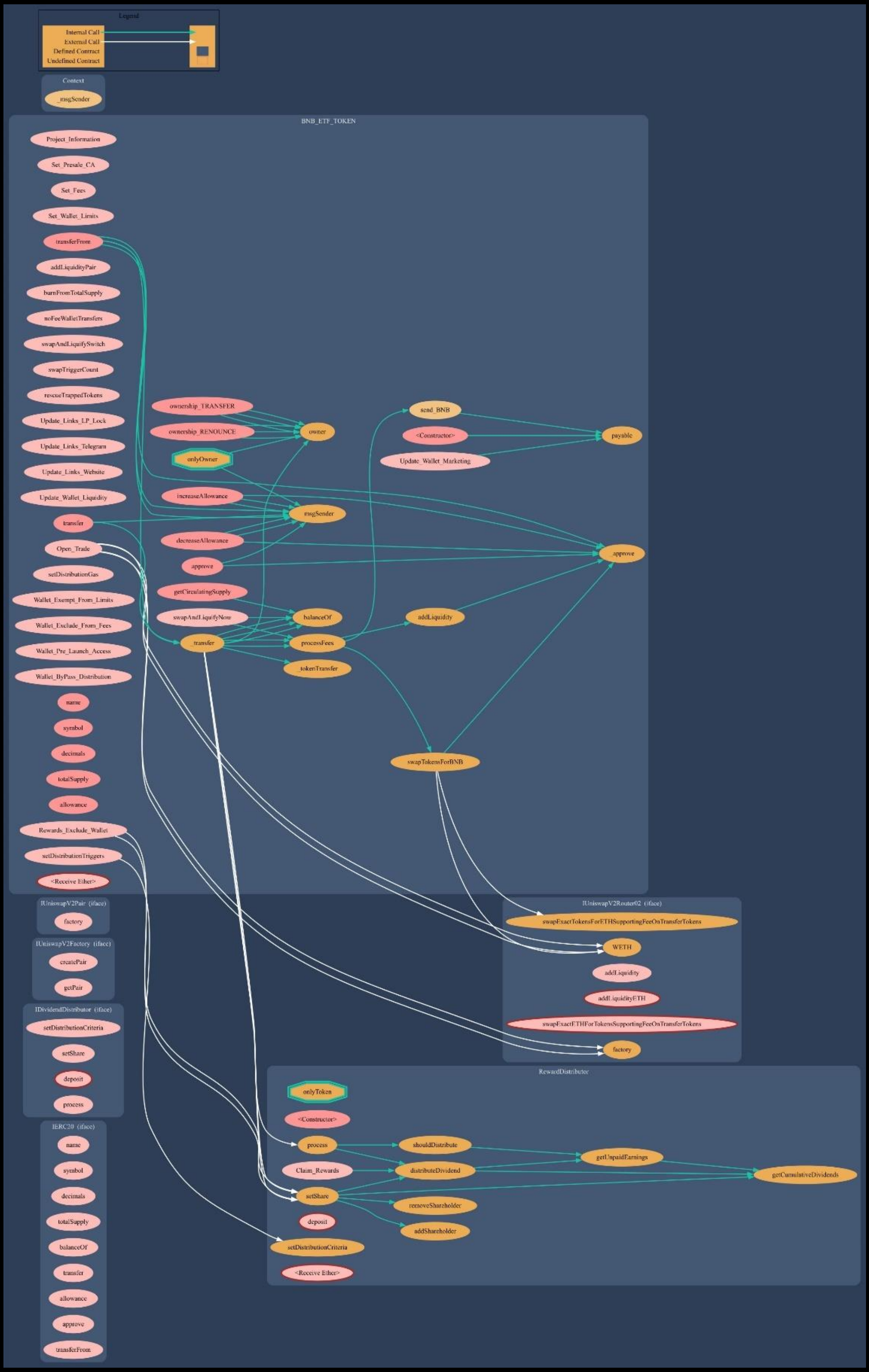
```
        ftrace | funcSig
987     function deposit() external payable override onlyToken {
988
989         uint256 balanceBefore = IERC20(RWDTKN).balanceOf(address(this));
990
991         address[] memory path = new address[](2);
992         path[0] = WBNB;
993         path[1] = address(RWDTKN);
994
995         DivRouter.swapExactETHForTokensSupportingFeeOnTransferTokens{value: msg.value}(
996             0,
997             path,
998             address(this),
999             block.timestamp
1000        );
1001
1002        uint256 amount = IERC20(RWDTKN).balanceOf(address(this)) - balanceBefore;
1003        totalDividends += amount;
1004        dividendsPerShare = dividendsPerShare + (dividendsPerShareAccuracyFactor * amount / totalShares);
1005
1006    }
1007
```

Outdated Compiler Version.

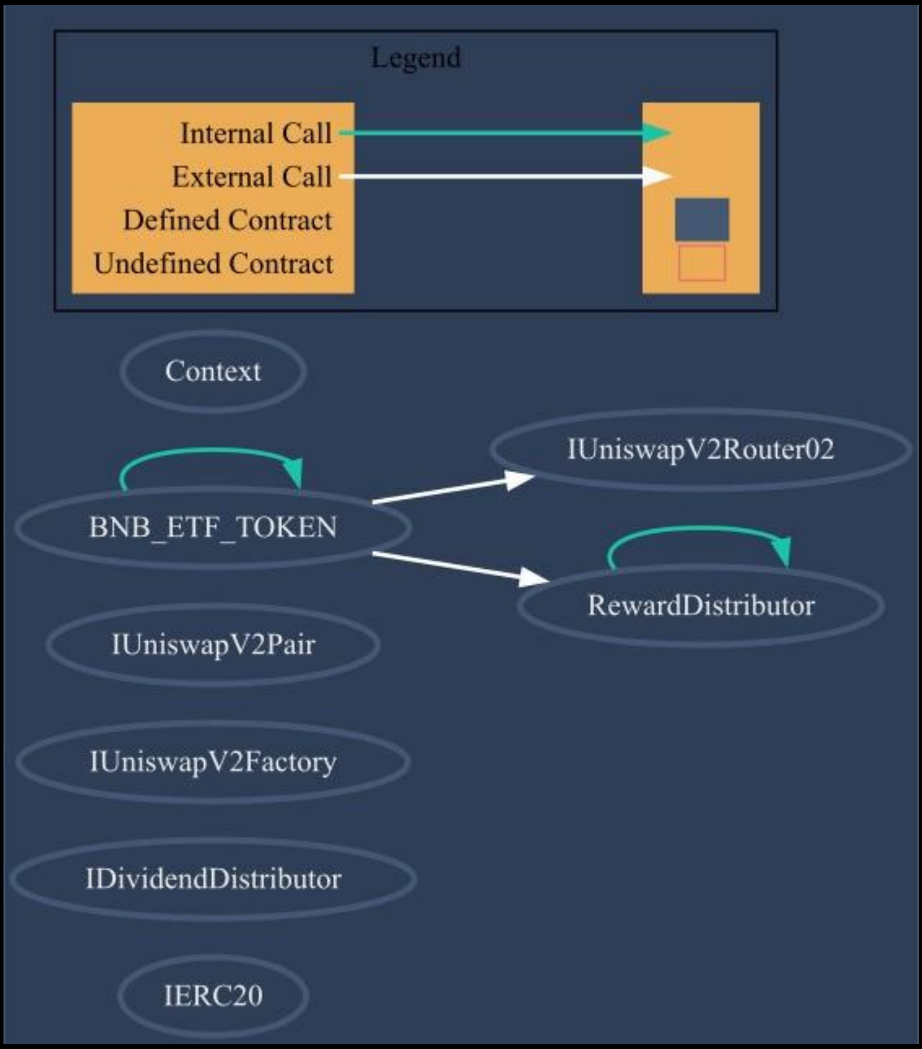
Item: 1	Location:	Line 6	Severity:	🟩 Low

Function	Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. The following outdated versions were detected: /bntf.sol - 0.8.19
Remedation	It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version v0.8.23, which patches most solidity vulnerabilities.

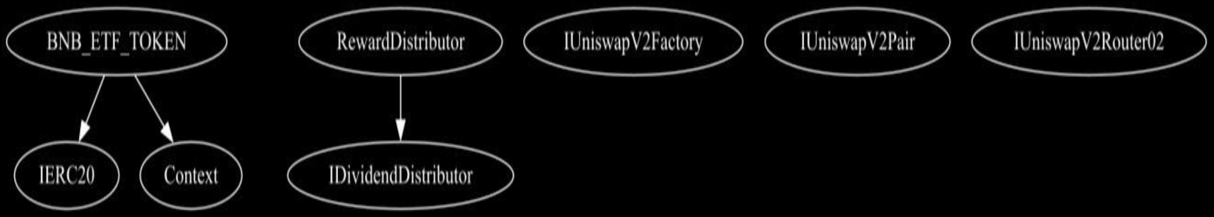
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
└	Function Name	Visibility	Mutability	Modifiers
IERC20	Interface			
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
IDividendDistributor	Interface			
└	setDistributionCriteria	External ❗	🛑	NO❗
└	setShare	External ❗	🛑	NO❗
└	deposit	External ❗	💵	NO❗
└	process	External ❗	🛑	NO❗
IUniswapV2Factory	Interface			
└	createPair	External ❗	🛑	NO❗
└	getPair	External ❗		NO❗
IUniswapV2Pair	Interface			
└	factory	External ❗		NO❗

Contract	Type	Bases		
IUniswapV2Router02	Interface			
└	factory	External ❗		NO❗
└	WETH	External ❗		NO❗
└	addLiquidity	External ❗	🛑	NO❗
└	addLiquidityETH	External ❗	💵	NO❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Context	Implementation			
└	_msgSender	Internal 🔒		
BNB_ETF_TOKEN	Implementation	Context, IERC20		
└		Public ❗	🛑	NO❗
└	Project_Information	External ❗		NO❗
└	Set_Presale_CA	External ❗	🛑	onlyOwner
└	Set_Fees	External ❗	🛑	onlyOwner
└	Set_Wallet_Limits	External ❗	🛑	onlyOwner
└	Open_Trade	External ❗	🛑	onlyOwner
└	addLiquidityPair	External ❗	🛑	onlyOwner
└	burnFromTotalSupply	External ❗	🛑	onlyOwner
└	noFeeWalletTransfers	External ❗	🛑	onlyOwner
└	swapAndLiquifySwitch	External ❗	🛑	onlyOwner

Contract	Type	Bases		
└	swapTriggerCount	External ▯	🛑	onlyOwner
└	swapAndLiquifyNow	External ▯	🛑	onlyOwner
└	rescueTrappedTokens	External ▯	🛑	onlyOwner
└	Update_Links_LP_Lock	External ▯	🛑	onlyOwner
└	Update_Links_Telegram	External ▯	🛑	onlyOwner
└	Update_Links_Website	External ▯	🛑	onlyOwner
└	Update_Wallet_Liquidity	External ▯	🛑	onlyOwner
└	Update_Wallet_Marketing	External ▯	🛑	onlyOwner
└	Rewards_Exclude_Wallet	External ▯	🛑	onlyOwner
└	setDistributionTriggers	External ▯	🛑	onlyOwner
└	setDistributionGas	External ▯	🛑	onlyOwner
└	Wallet_Exempt_From_Limits	External ▯	🛑	onlyOwner
└	Wallet_Exclude_From_Fees	External ▯	🛑	onlyOwner
└	Wallet_Pre_Launch_Access	External ▯	🛑	onlyOwner
└	Wallet_ByPass_Distribution	External ▯	🛑	onlyOwner
└	ownership_RENOUNCE	Public ▯	🛑	onlyOwner
└	ownership_TRANSFER	Public ▯	🛑	onlyOwner
└	owner	Public ▯		NO▯
└	name	Public ▯		NO▯

Contract	Type	Bases		
└	symbol	Public ❗️		NO❗️
└	decimals	Public ❗️		NO❗️
└	totalSupply	Public ❗️		NO❗️
└	balanceOf	Public ❗️		NO❗️
└	allowance	Public ❗️		NO❗️
└	increaseAllowance	Public ❗️	🛑	NO❗️
└	decreaseAllowance	Public ❗️	🛑	NO❗️
└	approve	Public ❗️	🛑	NO❗️
└	_approve	Private 🔐	🛑	
└	transfer	Public ❗️	🛑	NO❗️
└	transferFrom	Public ❗️	🛑	NO❗️
└	send_BNB	Internal 🔒	🛑	
└	getCirculatingSupply	Public ❗️		NO❗️
└	_transfer	Private 🔐	🛑	
└	processFees	Private 🔐	🛑	
└	swapTokensForBNB	Private 🔐	🛑	
└	addLiquidity	Private 🔐	🛑	
└	_tokenTransfer	Private 🔐	🛑	
└		External ❗️	💵	NO❗️
RewardDistributor	Implementation	IDividendDistributor		
└		Public ❗️	🛑	NO❗️
└	Claim_Rewards	External ❗️	🛑	NO❗️
└	setDistributionCriteria	External ❗️	🛑	onlyToken

Contract	Type	Bases		
└	setShare	External ❗	🛑	onlyToken
└	deposit	External ❗	💵	onlyToken
└	process	External ❗	🛑	onlyToken
└	shouldDistribute	Internal 🔒		
└	distributeDividend	Internal 🔒	🛑	
└	getUnpaidEarnings	Public ❗		NO❗
└	getCumulativeDividends	Internal 🔒		
└	addShareholder	Internal 🔒	🛑	
└	removeShareholder	Internal 🔒	🛑	
└		External ❗	💵	NO❗

🛑	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