SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Bad Cook

BAD

BEP20

0x211eb9732dc2694d7f59d9d81ecf047d90b8ea

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	Bad_Cook
Ticker/Simbol	BAD
Blockchain	Binance Smart Chain BEP20
Contract Address	0x211eb9732dc2694d7f59d9d81ecf047d90b8eab6
Creator Address	0x340d72c3492979fCd20f242C8b1f08DB1847FFE5
Current Owner Address	0x0000000000000000000000000000000000000000
Contract Explorer	https://bscscan.com/address/0x211eB9732DC2694d7F59D9D81ECf047d90b8eab6#code
Compiler Version	v0.8.19+commit.7dd6d404
License	None
Optimisation	Yes with 200 Runs
Total Supply	100,000 **BAD**
Decimals	4

Creation/Audit

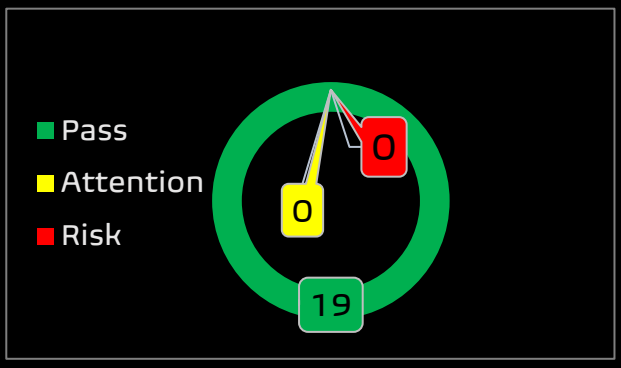
Contract Deployed	13.03.2024
Audit Created	25.03.2024
Audit Update	V 1.0

Verified Socials

Website	https://bad.cook.army
Telegram	https://t.me/BadCookArmy
Twitter (X)	https://twitter.com/BadCookArmy

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

BAD COOK 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		0x0000000000000000000000000000000000000000 Sometimes referred to as the "zero address" or "dead address" and is not owned by anyone.
Buy Tax	0 %	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	0 %	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 24.03.2024 95% for 9414 days on Unicrypt
Trading Disable Functions	✓	No Trading suspendable function found. If a suspendable code is included, the token maybe neither be bought or sold (honeypot risk). If contract is renounced this function can't be used
Set Fees function	✓	No Fee Setting function found The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk).
Proxy Contract	✓	Not a Proxy contract.
Mint Function	✓	No Mint Function detected Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token. Owner can mint new tokens and sell.

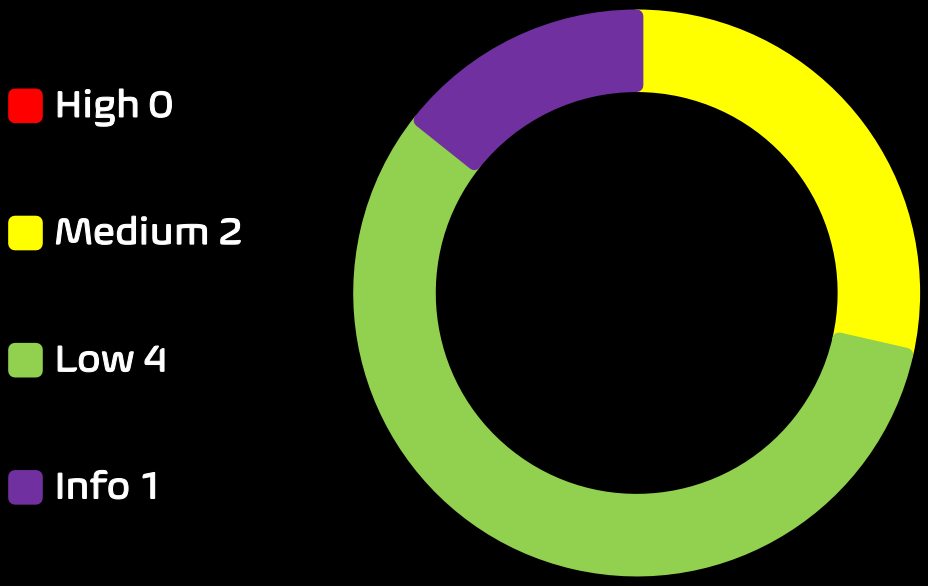
Balance Modifier Function	✅	No Balance Modifier function found. If there is a function for this, the contract owner can have the authority to modify the balance of tokens at other addresses. For example revoke the bought tokens from the holders wallet. Common form of scam: You buy the token, but it's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

Removing Risk of contract function based on renounced ownership

Contract Security

Total Findings: 7



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)

- Approve Front Running Attack (Sandwich Bot Attack)
- Public Burn Function

Low severity issues: (4)

- Missing Events
- Long number literals
- Outdated Compiler Version
- Floating Pragma

Informational severity issues: (1)

- Public Functions Should be Declared External

Contract Weakness Classisication

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

ID	Description	AI	Manual	Result
SWC-100	Function Default Visibility	Passed	Passed	Passed
SWC-101	Integer Overflow and Underflow	Passed	Passed	Passed
SWC-102	Outdated Compiler Version	low	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	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. (3 Item)

Item: 1	Location:	Line erc20.sol 136-140	Severity:	Medium

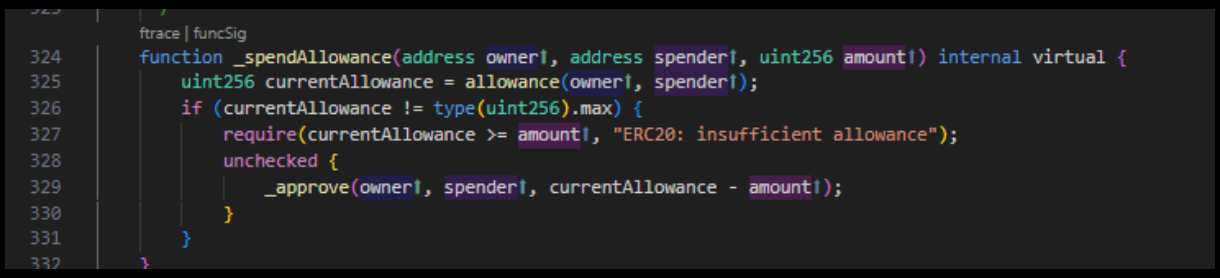
Function	The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when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

```
ftrace | funcSig
function _approve(address owner, address spender, uint256 amount) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

Item: 2	Location:	Line erc20.sol 324-332	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 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
function _spendAllowance(address owner1, address spender1, uint256 amount1) internal virtual {
    uint256 currentAllowance = allowance(owner1, spender1);
    if (currentAllowance != type(uint256).max) {
        require(currentAllowance >= amount1, "ERC20: insufficient allowance");
        unchecked {
            _approve(owner1, spender1, currentAllowance - amount1);
        }
    }
}
```

Item: 3	Location:	Line erc20permit.sol 49-68	Severity:	Medium

Function	The permit() 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 permit 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
49    function permit(
50        address owner,
51        address spender,
52        uint256 value,
53        uint256 deadline,
54        uint8 v,
55        bytes32 r,
56        bytes32 s
57    ) public virtual override {
58        require(block.timestamp <= deadline, "ERC20Permit: expired deadline");
59
60        bytes32 structHash = keccak256(abi.encode(_PERMIT_TYPEHASH, owner, spender, value, _useNonce(owner), deadline));
61
62        bytes32 hash = _hashTypedDataV4(structHash);
63
64        address signer = ECDSA.recover(hash, v, r, s);
65        require(signer == owner, "ERC20Permit: invalid signature");
66
67        _approve(owner, spender, value);
68    }
69
70    /**
```

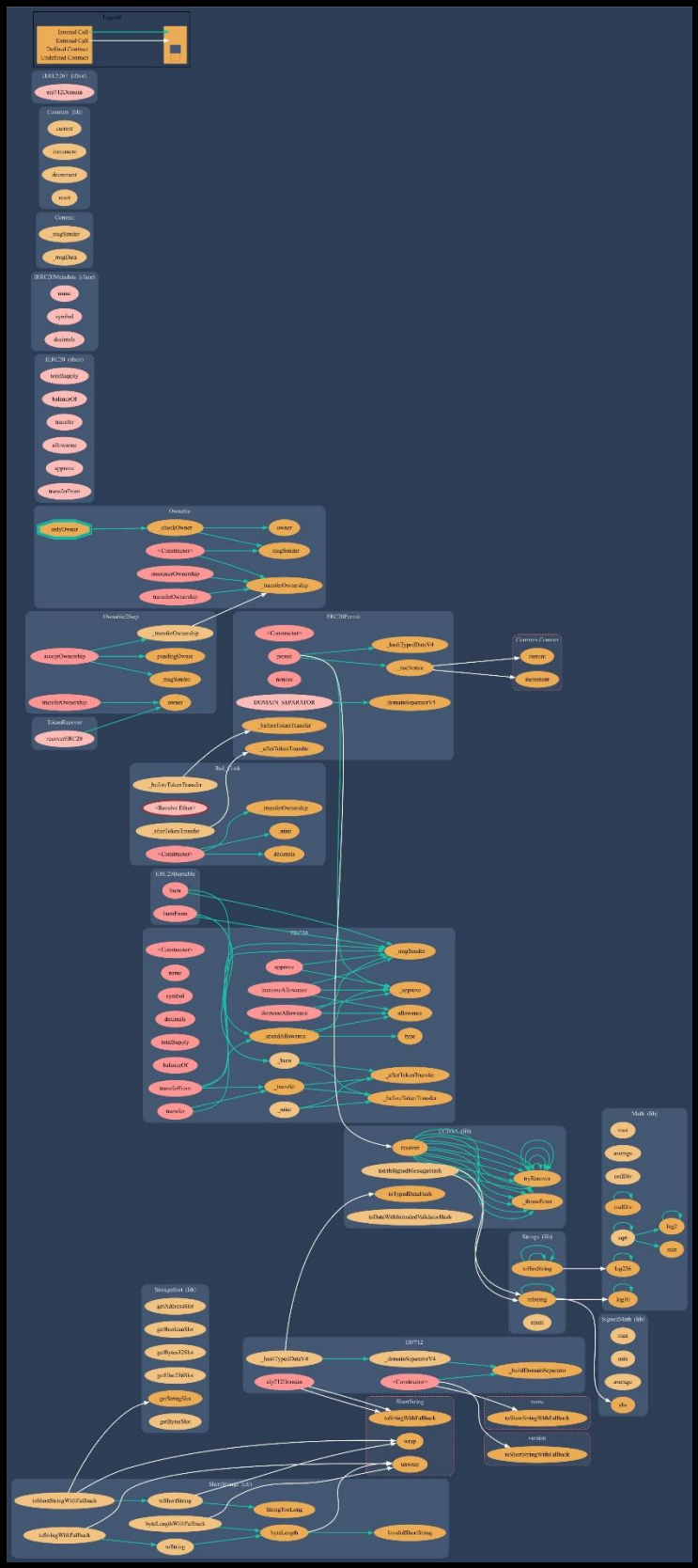
⚠ Public Burn function

Item: 1	Location:	Line erc20burnable.sol 35-49	Severity:	Medium

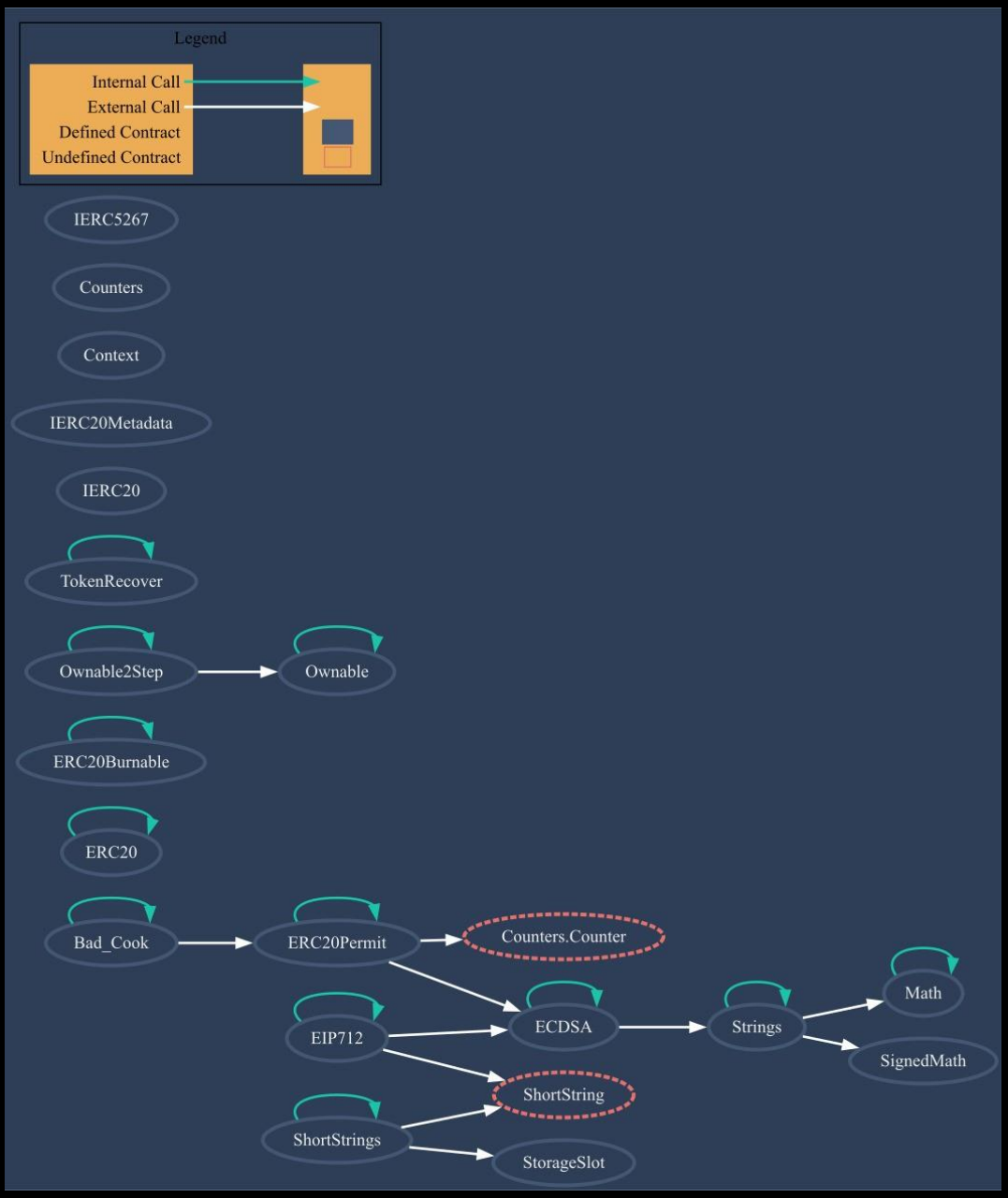
Function	The contract was found to be using public or an external burn function. The function was missing access control to prevent another user from burning their tokens. Also, the burn function was found to be using a different address than msg.sender. The burn function is public, allowing any user to burn tokens. An attacker could exploit this to manipulate token supply, potentially leading to price inflation and draining liquidity pools.
Remedation	1. Ensure that initialization functions can only be called once and only by authorized entities. 2. Implement least-privilege roles using libraries like OpenZeppelin's Access Control. 3. Add proper access control modifiers to sensitive functions, such as onlyOwner or custom roles.

```
ftrace | funcSig
35    function burnFrom(address account1, uint256 amount1) public virtual {
36        _spendAllowance(account1, _msgSender(), amount1);
37        _burn(account1, amount1);
38    }
39 }
```

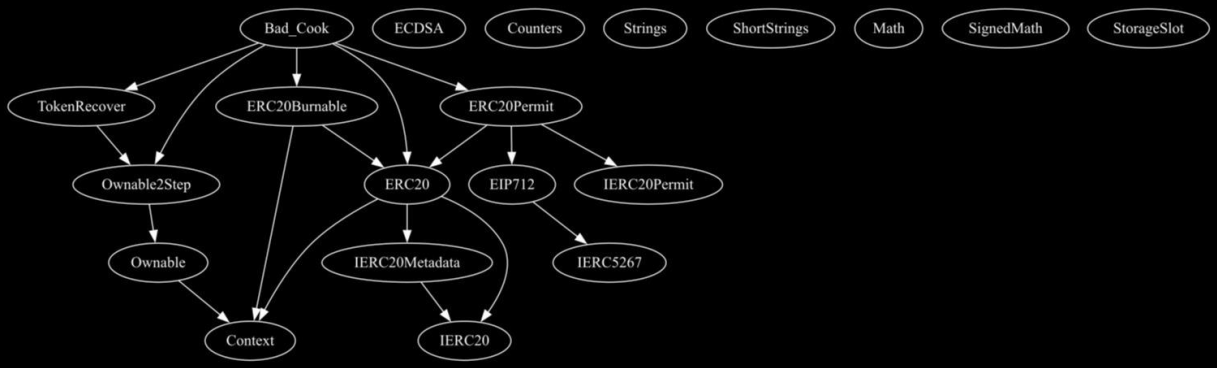
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
Dogacoin	Implementation	ERC20, ERC20Burnable, Ownable2Step, DividendTracker Functions, Initializable		
L		Public ❗	🛑	ERC20
L	initialize	External ❗	🛑	initializer
L		External ❗	💵	NO❗
L	decimals	Public ❗		NO❗
L	_swapTokensForCoin	Private 🔐	🛑	
L	updateSwapThreshold	Public ❗	🛑	onlyOwner
L	getSwapThresholdAmount	Public ❗		NO❗
L	getAllPending	Public ❗		NO❗
L	dogacoinmarketingAddressSetup	Public ❗	🛑	onlyOwner
L	dogacoinmarketingFeesSetup	Public ❗	🛑	onlyOwner
L	_swapAndLiquify	Private 🔐	🛑	
L	_addLiquidity	Private 🔐	🛑	
L	addLiquidityFromLeftoverTokens	External ❗	🛑	NO❗
L	liquidityFeesSetup	Public ❗	🛑	onlyOwner

L	_swapTokensForOtherRewardTokens	Private		
L	_sendDividends	Private		
L	excludeFromDividends	External		onlyOwner
L	_excludeFromDividends	Internal		
L	rewardsFeesSetup	Public		onlyOwner
L	_burn	Internal		
L	_mint	Internal		
L	excludeFromFees	Public		onlyOwner
L	_transfer	Internal		
L	_updateRouterV2	Private		
L	setAMMPair	External		onlyOwner
L	_setAMMPair	Private		
L	excludeFromLimits	External		onlyOwner
L	_excludeFromLimits	Internal		
L	updateMaxWalletAmount	Public		onlyOwner
L	_maxWalletSafeLimit	Private		
L	_maxTxSafeLimit	Private		
L	updateMaxBuyAmount	Public		onlyOwner
L	updateMaxSellAmount	Public		onlyOwner

L	enableTrading	External ❗	🔘	onlyOwner
L	excludeFromTradingRestriction	Public ❗	🔘	onlyOwner
L	_beforeTokenTransfer	Internal 🔒	🔘	
L	_afterTokenTransfer	Internal 🔒	🔘	
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	_burn	Internal 🔒	🔘	
L	_approve	Internal 🔒	🔘	

L	_spendAllowance	Internal 🔒	🔘	
L	_beforeTokenTransfer	Internal 🔒	🔘	
L	_afterTokenTransfer	Internal 🔒	🔘	
ERC20Burnable	Implementation	Context, ERC20		
L	burn	Public ❗	🔘	NO❗
L	burnFrom	Public ❗	🔘	NO❗
Ownable2Step	Implementation	Ownable		
L	pendingOwner	Public ❗		NO❗
L	transferOwnership	Public ❗	🔘	onlyOwner
L	_transferOwnership	Internal 🔒	🔘	
L	acceptOwnership	Public ❗	🔘	NO❗
SafeMathUint	Library			
L	toInt256Safe	Internal 🔒		
SafeMathInt	Library			
L	toUint256Safe	Internal 🔒		
TruncatedERC20	Implementation			
L		Public ❗	🔘	NO❗
L	name	Public ❗		NO❗
L	symbol	Public ❗		NO❗
L	decimals	Public ❗		NO❗

L	totalSupply	Public ❗		NO❗
L	balanceOf	Public ❗		NO❗
L	_mint	Internal 🔒	🛑	
L	_burn	Internal 🔒	🛑	
DividendPayingTokenInterface	Interface			
L	dividendOf	External ❗		NO❗
DividendPayingTokenOptionalInterface	Interface			
L	withdrawableDividendOf	External ❗		NO❗
L	withdrawnDividendOf	External ❗		NO❗
L	accumulativeDividendOf	External ❗		NO❗
DividendPayingToken	Implementation	TruncatedERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
L		Public ❗	🛑	TruncatedERC20
L	distributeDividends	Public ❗	🛑	NO❗
L	_withdrawDividend	Internal 🔒	🛑	
L	dividendOf	Public ❗		NO❗
L	withdrawableDividendOf	Public ❗		NO❗
L	withdrawnDividendOf	Public ❗		NO❗

L	accumulativeDividendOf	Public ❗		NO❗
L	_mint	Internal 🔒	🔘	
L	_burn	Internal 🔒	🔘	
L	_setBalance	Internal 🔒	🔘	
IterableMapping	Library			
L	get	Public ❗		NO❗
L	getIndexOfKey	Public ❗		NO❗
L	getKeyAtIndex	Public ❗		NO❗
L	size	Public ❗		NO❗
L	set	Public ❗	🔘	NO❗
L	remove	Public ❗	🔘	NO❗
DividendTracker	Implementation	Ownable, DividendPayingToken		
L		Public ❗	🔘	DividendPayingToken
L	setRewardToken	External ❗	🔘	onlyOwner
L	excludeFromDividends	External ❗	🔘	onlyOwner
L	claimWaitSetup	Public ❗	🔘	onlyOwner
L	getNumberOfTokenHolders	External ❗		NO❗
L	getAccountData	Public ❗		NO❗
L	getAccountDataAtIndex	Public ❗		NO❗
L	claim	Public ❗	🔘	onlyOwner
L	_canAutoClaim	Private 🔐		

L	setBalance	Public ❗	🔘	onlyOwner
L	process	External ❗	🔘	onlyOwner
DividendTracker Functions	Implementation	Ownable2Step		
L	_deployDividend Tracker	Internal 🔒	🔘	
L	_setRewardToke n	Internal 🔒	🔘	
L	gasForProcessin gSetup	Public ❗	🔘	onlyOwner
L	claimWaitSetup	External ❗	🔘	onlyOwner
L	_excludeFromDi vidends	Internal 🔒	🔘	
L	isExcludedFrom Dividends	Public ❗		NO❗
L	claim	External ❗	🔘	NO❗
L	getClaimWait	External ❗		NO❗
L	getTotalDividen dsDistributed	External ❗		NO❗
L	withdrawableDi videndOf	Public ❗		NO❗
L	dividendTokenB alanceOf	Public ❗		NO❗
L	dividendTokenT otalSupply	Public ❗		NO❗
L	getAccountDivid endsInfo	External ❗		NO❗
L	getAccountDivid endsInfoAtIndex	External ❗		NO❗
L	getLastProcesse dIndex	External ❗		NO❗

L	getNumberOfDividendTokenHolders	Public ❗		NO❗
L	process	External ❗	🔘	NO❗
Initializable	Implementation			
IUniswapV2Factory	Interface			
L	feeTo	External ❗		NO❗
L	feeToSetter	External ❗		NO❗
L	getPair	External ❗		NO❗
L	allPairs	External ❗		NO❗
L	allPairsLength	External ❗		NO❗
L	createPair	External ❗	🔘	NO❗
L	setFeeTo	External ❗	🔘	NO❗
L	setFeeToSetter	External ❗	🔘	NO❗
IUniswapV2Pair	Interface			
L	name	External ❗		NO❗
L	symbol	External ❗		NO❗
L	decimals	External ❗		NO❗
L	totalSupply	External ❗		NO❗
L	balanceOf	External ❗		NO❗
L	allowance	External ❗		NO❗
L	approve	External ❗	🔘	NO❗
L	transfer	External ❗	🔘	NO❗
L	transferFrom	External ❗	🔘	NO❗

L	DOMAIN_SEPARATOR	External ❗		NO❗
L	PERMIT_TYPEHASH	External ❗		NO❗
L	nonces	External ❗		NO❗
L	permit	External ❗	🔘	NO❗
L	MINIMUM_LIQUIDITY	External ❗		NO❗
L	factory	External ❗		NO❗
L	token0	External ❗		NO❗
L	token1	External ❗		NO❗
L	getReserves	External ❗		NO❗
L	price0CumulativeLast	External ❗		NO❗
L	price1CumulativeLast	External ❗		NO❗
L	kLast	External ❗		NO❗
L	mint	External ❗	🔘	NO❗
L	burn	External ❗	🔘	NO❗
L	swap	External ❗	🔘	NO❗
L	skim	External ❗	🔘	NO❗
L	sync	External ❗	🔘	NO❗
L	initialize	External ❗	🔘	NO❗
IUniswapV2Router01	Interface			
L	factory	External ❗		NO❗
L	WETH	External ❗		NO❗
L	addLiquidity	External ❗	🔘	NO❗

L	addLiquidityETH	External ❗	💸	NO❗
L	removeLiquidity	External ❗	🛑	NO❗
L	removeLiquidity ETH	External ❗	🛑	NO❗
L	removeLiquidity WithPermit	External ❗	🛑	NO❗
L	removeLiquidity ETHWithPermit	External ❗	🛑	NO❗
L	swapExactToke nsForTokens	External ❗	🛑	NO❗
L	swapTokensFor ExactTokens	External ❗	🛑	NO❗
L	swapExactETHF orTokens	External ❗	💸	NO❗
L	swapTokensFor ExactETH	External ❗	🛑	NO❗
L	swapExactToke nsForETH	External ❗	🛑	NO❗
L	swapETHForExa ctTokens	External ❗	💸	NO❗
L	quote	External ❗		NO❗
L	getAmountOut	External ❗		NO❗
L	getAmountIn	External ❗		NO❗
L	getAmountsOut	External ❗		NO❗
L	getAmountsIn	External ❗		NO❗
IUniswapV2Rout er02	Interface	IUniswapV2Rout er01		
L	removeLiquidity ETHSupportingF eeOnTransferTo kens	External ❗	🛑	NO❗
L	removeLiquidity ETHWithPermit SupportingFeeO	External ❗	🛑	NO❗

	nTransferTokens			
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Ownable	Implementation	Context		
L		Public ❗	🛑	NO❗
L	owner	Public ❗		NO❗
L	_checkOwner	Internal 🔒		
L	renounceOwnership	Public ❗	🛑	onlyOwner
L	transferOwnership	Public ❗	🛑	onlyOwner
L	_transferOwnership	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❗
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	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