SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Phoenix Token

PHT

BEP20

0x885c99a787BE6b41cbf964174C771A9f7ec48

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	PhoenixToken
Ticker/Simbol	PHT
Blockchain	Binance Smart Chain BEP20
Contract Address	0x885c99a787BE6b41cbf964174C771A9f7ec48e04
Creator Address	0xF73CDec6497a643332a88A0cECaC0CAdD1E998B7
Current Owner Address	0x0000000000000000000000000000000000000000
Contract Explorer	https://bscscan.com/address/0x885c99a787BE6b41cbf964174C771A9f7ec48e04#code
Compiler Version	**v0.8.15+commit.e14f2714**
License	MIT
Optimisation	Yes with 200 Runs
Total Supply	9,829,617.835952 **PHT**
Decimals	18

Creation/Audit

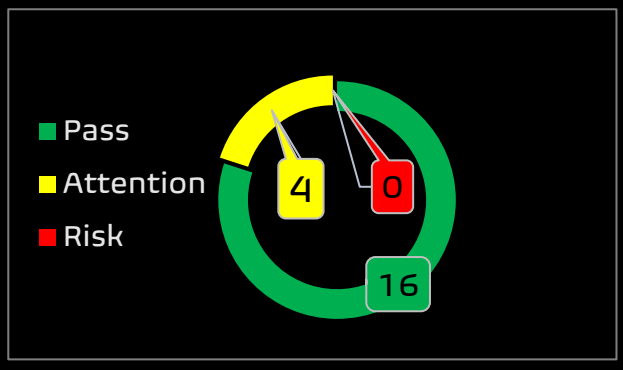
Contract Deployed	26.11.2023
Audit Created	25.06.2024
Audit Update	V 1.0

Verified Socials

Website	https://phoenixtoken.community/
Telegram	https://t.me/phoenixtoken0
Twitter (X)	https://twitter.com/PhoenixToken0/

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
Buy Tax	5 %	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	5 %	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 25.06.2024 92,9% Locked for 150 Days on Mudra Locker
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. Contract renounced, function can not be triggered by owner 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 and Multi-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. Contract renounced, function can not be triggered by owner 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. Contract renounced, function can not be triggered by owner. 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. Contract renounced, function can not be triggered by owner 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

Transaction Receipt Event Logs

345

Address Phoenix Token: PHT Token

Name `OwnershipTransferred (index_topic_1 address previousOwner, index_topic_2 address newOwner)` View Source

Topics

0 `0x8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0`

1: previousOwner Dec → `0xF73CDec6497a643332a88A0cECaC0CAdD1E998B7`

2: newOwner Dec → `0x0000000000000000000000000000000000000000`

Data `0x`

Following detected contract functions serve as informational purposes about the contract. The owner has no more authorisation to trigger the following functions.

Set Fee

Contract renounced, function can not be triggered by owner

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 updateFees(
    uint256 deadBuy,
    uint256 deadSell,
    uint256 marketingBuy,
    uint256 marketingSell,
    uint256 liquidityBuy,
    uint256 liquiditySell,
    uint256 RewardsBuy,
    uint256 RewardsSell,
    uint256 devBuy,
    uint256 devSell
) public onlyOwner {
    buyDeadFees = deadBuy;
    buyMarketingFees = marketingBuy;
    buyLiquidityFee = liquidityBuy;
    buyRewardsFee = RewardsBuy;
    sellDeadFees = deadSell;
    sellMarketingFees = marketingSell;
    sellLiquidityFee = liquiditySell;
    sellRewardsFee = RewardsSell;
    buyDevFee = devBuy;
    sellDevFee = devSell;

    totalSellFees = sellRewardsFee
        .add(sellLiquidityFee)
        .add(sellMarketingFees)
        .add(sellDevFee);

    totalBuyFees = buyRewardsFee
        .add(buyLiquidityFee)
        .add(buyMarketingFees)
        .add(buyDevFee);

    require(totalSellFees <= 100 && totalBuyFees <= 100, "total fees cannot exceed 15% sell or buy");

    emit UpdateFees(
        sellDeadFees,
        sellMarketingFees,
        sellLiquidityFee,
        sellRewardsFee,
        buyDeadFees,
        buyMarketingFees,
        buyLiquidityFee,
        buyRewardsFee,
        buyDevFee,
        sellDevFee
    );
```

Whitelist (Set wallets excluded from fees)

Contract renounced, function can not be triggered by owner

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)

```
// exclude a wallet from fees
ftrace | funcSig
function setExcludeFees(address account, bool excluded) public onlyOwner {
    _isExcludedFromFees[account] = excluded;
    emit ExcludeFromFees(account, excluded);
}
```

⚠ Max Transaction and Holding Modify function

Contract renounced, function can not be triggered by owner

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

```
// set max wallet, can not be lower than 0.05% of supply
ftrace | funcSig
function setmaxWallet(uint256 value) external onlyOwner {
    value = value * (10**18);
    require(value >= _totalSupply / 2000, "max wallet cannot be set to less than 0.05%");
    maxWallet = value;
}
```

⚠ Blacklist Wallets from receiving dividend

Contract renounced, function can not be triggered by owner

Wallets can be blacklisted from receiving dividens rewards. In some cases this is to avoid team wallets, burn address to receive rewards, in some cases holder wallets can be blacklisted as well.

```
ftrace | funcSig
function setExcludeDividends(address account) public onlyOwner {
    dividendTracker.excludeFromDividends(account);
}
```

⚠ Trading Cooldown (max 300 sec)

Contract renounced, function can not be triggered by owner

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.

```
// set cooldown timer, can only be between 0 and 300 seconds (5 mins max)
ftrace | funcSig
function setcooldowntimer(uint256 value) external onlyOwner {
    require(value <= 300, "cooldown timer cannot exceed 5 minutes");
    cooldowntimer = value;
}
```

⚠ Update Transfer Fee

Contract renounced, function can not be triggered by owner

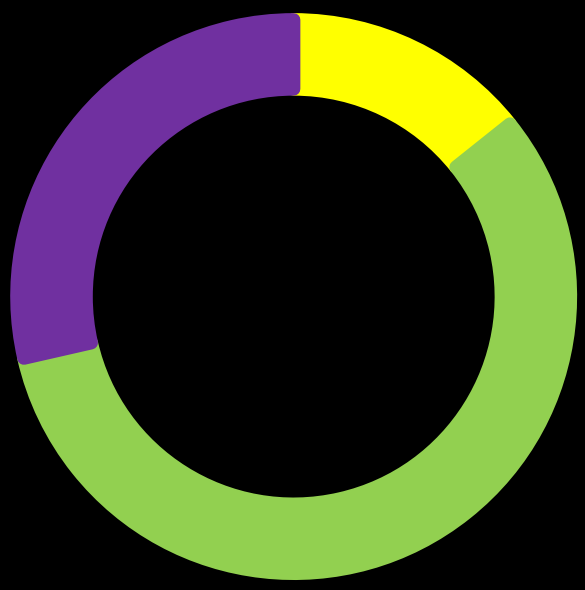
The transfer fee can be updated. This means transfer between wallets gets taxed.

```
function updateTransferFee(uint256 newTransferFee) public onlyOwner {
    require (newTransferFee <= 15, "transfer fee cannot exceed 15%");
    transferFee = newTransferFee;
    emit UpdateTransferFee(transferFee);
}
```

Contract Security

Total Findings: 9

- High 0
- Medium 1
- Low 4
- 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 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

High severity Issues: (0)

Medium severity issues: (1)

- Usage of tx. origin

Low severity issues: (4)

- Missing Events
- Long number literals
- Outdated compiler Version
- Approve of Front Running Attack

Informational severity issues: (2)

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

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	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	High	Medium	Medium
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 (2 Item)

Item: 1	Location:	Line 270-278	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

```
        ↑trace | funcSig
270     function approve(address spender↑, uint256 amount↑)
271         public
272         virtual
273         override
274         returns (bool)
275     {
276         _approve(_msgSender(), spender↑, amount↑);
277         return true;
278     }
279
```

Item: 2	Location:	Line 1625-1632	Severity:	Low

Function	The swapTokensForEth() 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 swapTokensForEth 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

```
1625    function swapTokensForEth(uint256 tokenAmount) private {
1626        address[] memory path = new address[](2);
1627        path[0] = address(this);
1628        path[1] = uniswapV2Router.WETH();
1629        _approve(address(this), address(uniswapV2Router), tokenAmount);
1630        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1631            tokenAmount,
1632            0, // accept any amount of ETH
1633            path,
1634            address(this),
1635            block.timestamp
1636        );
1637    }
1638
```

⚠ Outdated Compiler Version. (18 Items)

Item: 1	Location:	Line 16	Severity:	Low

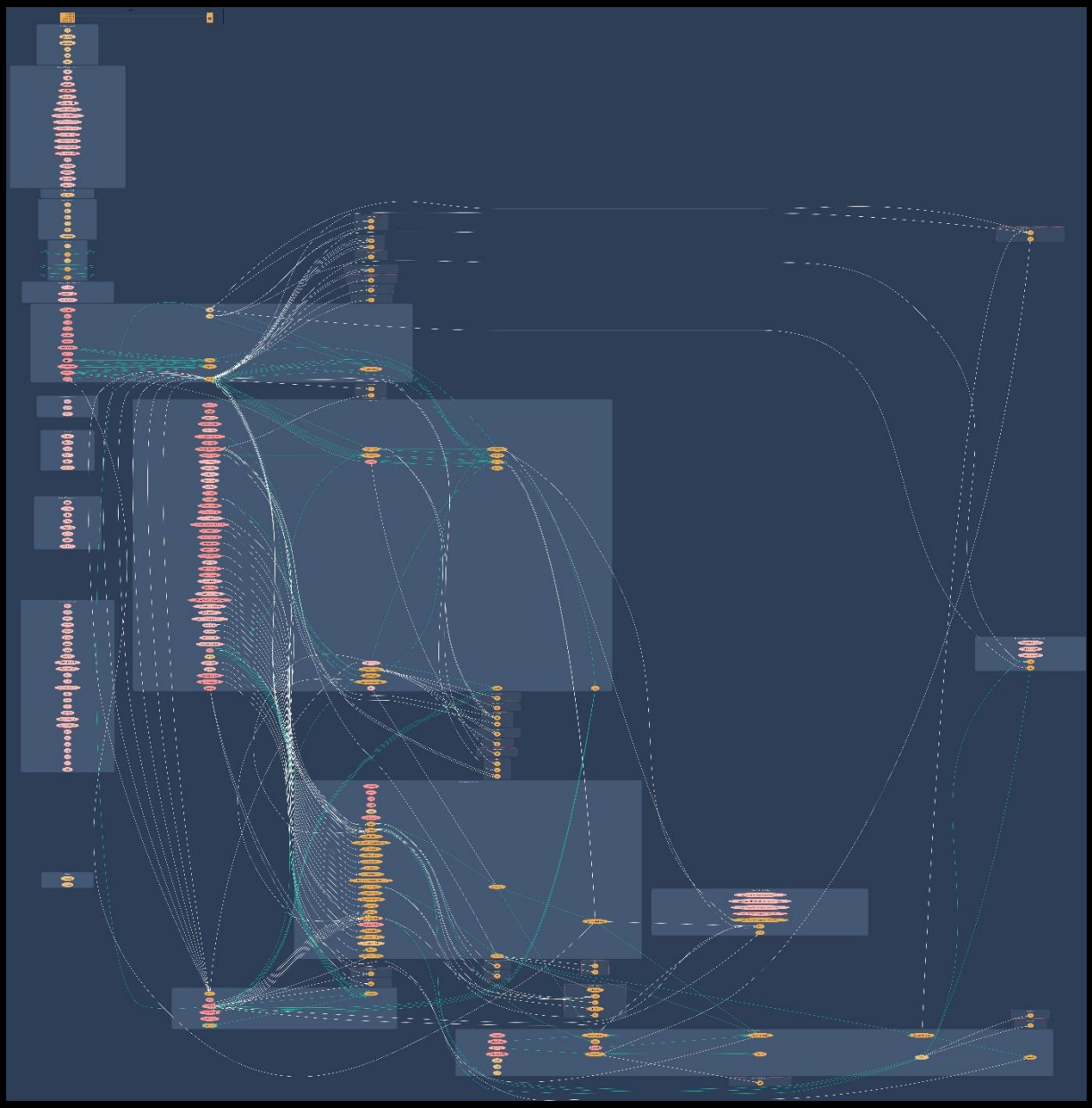
Function	Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. The following outdated versions were detected: /phoenix.sol - ^0.8.15
Remedation	It is recommended to use a recent version of the Solidity 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 v0.8.25, which patches most solidity vulnerabilities.

⚠ Usage of tx. origin (6 Item)

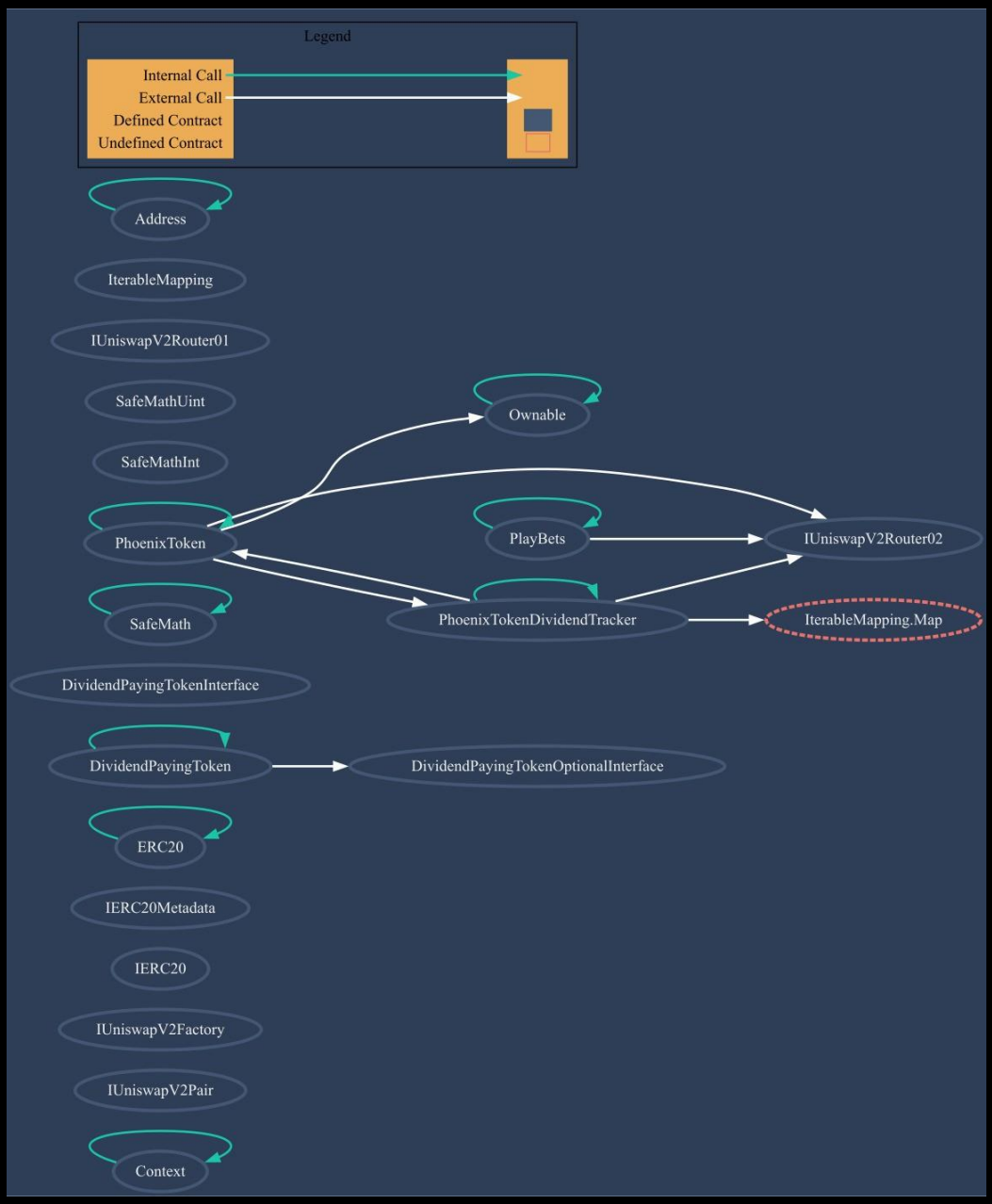
Item: 1	Location:	Limitless.sol Line 1405	Severity:	Medium
Item: 2	Location:	Limitless.sol Line 1499	Severity:	Medium
Item: 3	Location:	Limitless.sol Line 1501	Severity:	Medium
Item: 4	Location:	Limitless.sol Line 1515	Severity:	Medium
Item: 5	Location:	Limitless.sol Line 1516	Severity:	Medium
Item: 6	Location:	Limitless.sol Line 1596	Severity:	Medium

Function	In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable. For example, if an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.
Remedation	The best way to prevent Tx Origin attacks is not to use the tx.origin for authentication purposes. Instead, it is advisable to use msg.sender You can implement a require of the form require(tx.origin == msg.sender).

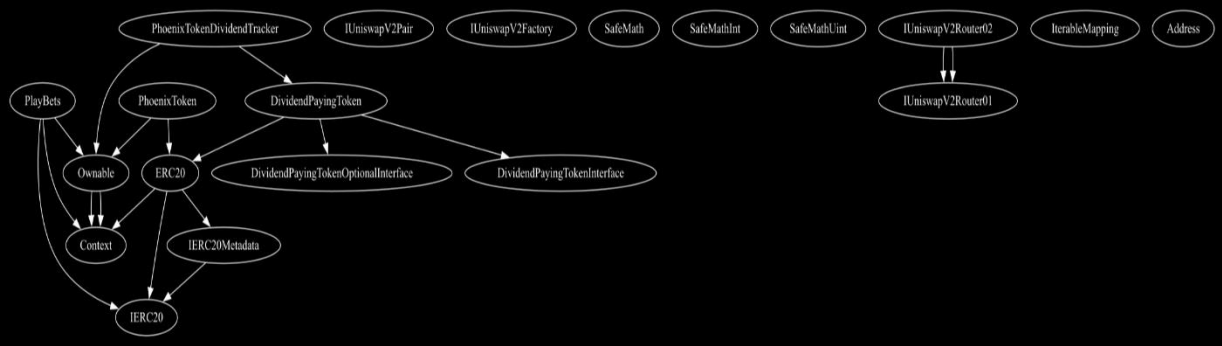
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
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❗

Contract	Type	Bases		
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!
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!
IERC20	Interface			
L	totalSupply	External !		NO!
L	balanceOf	External !		NO!
L	transfer	External !		NO!

Contract	Type	Bases		
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	_burn	Internal		

Contract	Type	Bases		
L	_approve	Internal 🔒		
L	_beforeTokenTransfer	Internal 🔒		
DividendPayingTokenOptionalInterface	Interface			
L	withdrawableDividendOf	External !		NO!
L	withdrawnDividendOf	External !		NO!
L	accumulativeDividendOf	External !		NO!
DividendPayingTokenInterface	Interface			
L	dividendOf	External !		NO!
L	distributeDividends	External !		NO!
L	withdrawDividend	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 🔒		
Ownable	Implementation	Context		
L		Public !		NO!

Contract	Type	Bases		
L	owner	Public		NO!
L	renounceOwnership	Public		onlyOwner
L	transferOwnership	Public		onlyOwner
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		
IUniswapV2Router01	Interface			
L	factory	External		NO!
L	WETH	External		NO!
L	addLiquidity	External		NO!
L	addLiquidityETH	External		NO!
L	removeLiquidity	External		NO!
L	removeLiquidityETH	External		NO!
L	removeLiquidityWithPermit	External		NO!
L	removeLiquidityETHWithPermit	External		NO!

Contract	Type	Bases		
L	swapExactTokens ForTokens	External !		NO!
L	swapTokensForEx actTokens	External !		NO!
L	swapExactETHFor Tokens	External !		NO!
L	swapTokensForEx actETH	External !		NO!
L	swapExactTokens ForETH	External !		NO!
L	swapETHForExact Tokens	External !		NO!
L	quote	External !		NO!
L	getAmountOut	External !		NO!
L	getAmountIn	External !		NO!
L	getAmountsOut	External !		NO!
L	getAmountsIn	External !		NO!
IUniswapV2Router 02	Interface	IUniswapV2Router 01		
L	removeLiquidityE THSupportingFee OnTransferTokens	External !		NO!
L	removeLiquidityE THWithPermitSup portingFeeOnTran sferTokens	External !		NO!
L	swapExactTokens ForTokensSupport ingFeeOnTransfer Tokens	External !		NO!
L	swapExactETHFor TokensSupporting FeeOnTransferTok ens	External !		NO!
L	swapExactTokens ForETHSupporting FeeOnTransferTok ens	External !		NO!

Contract	Type	Bases		
DividendPayingToken	Implementation	ERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
L		Public !		ERC20
L		External !		NO!
L	distributeDividends	Public !		NO!
L	withdrawDividend	Public !		NO!
L	_withdrawDividendOfUser	Internal		
L	dividendOf	Public !		NO!
L	withdrawableDividendOf	Public !		NO!
L	withdrawnDividendOf	Public !		NO!
L	accumulativeDividendOf	Public !		NO!
L	_transfer	Internal		
L	_mint	Internal		
L	_burn	Internal		
L	_setBalance	Internal		
PhoenixToken	Implementation	ERC20, Ownable		
L		Public !		ERC20
L	decimals	Public !		NO!
L		External !		NO!
L	updateStakingAmounts	Public !		onlyOwner
L	enableTrading	External !		onlyOwner

Contract	Type	Bases		
L	setPresaleWallet	External		onlyOwner
L	setExcludeFees	Public		onlyOwner
L	setExcludeDividends	Public		onlyOwner
L	setIncludeDividends	Public		onlyOwner
L	setCanTransferBefore	External		onlyOwner
L	setLimitsInEffect	External		onlyOwner
L	setGasPriceLimit	External		onlyOwner
L	setcooldowntimer	External		onlyOwner
L	setmaxWallet	External		onlyOwner
L	enableStaking	Public		onlyOwner
L	stake	Public		NO
L	setSwapTriggerAmount	Public		onlyOwner
L	enableSwapAndLiquify	Public		onlyOwner
L	setAutomatedMarketMakerPair	Public		onlyOwner
L	setAllowCustomTokens	Public		onlyOwner
L	setAllowAutoReinvest	Public		onlyOwner
L	_setAutomatedMarketMakerPair	Private		
L	updateGasForProcessing	Public		onlyOwner
L	transferAdmin	Public		onlyOwner
L	updateTransferFee	Public		onlyOwner
L	updateFees	Public		onlyOwner

Contract	Type	Bases		
L	getStakingInfo	External		NO!
L	getTotalDividends Distributed	External		NO!
L	isExcludedFromFe es	Public		NO!
L	withdrawableDivi dendOf	Public		NO!
L	dividendTokenBal anceOf	Public		NO!
L	getAccountDivide ndsInfo	External		NO!
L	getAccountDivide ndsInfoAtIndex	External		NO!
L	processDividendTr acker	External		NO!
L	claim	External		NO!
L	getLastProcessedI ndex	External		NO!
L	getNumberOfDivi dendTokenHolder s	External		NO!
L	setAutoClaim	External		NO!
L	setReinvest	External		NO!
L	setDividendsPaus ed	External		onlyOwner
L	isExcludedFromAu toClaim	External		NO!
L	isReinvest	External		NO!
L	_transfer	Internal		
L	getStakingBalance	Private		
L	swapAndLiquify	Private		
L	swapTokensForEt h	Private		

Contract	Type	Bases		
L	updatePayoutToken	Public		onlyOwner
L	getPayoutToken	Public		NO!
L	setMinimumTokenBalanceForAutoDividends	Public		onlyOwner
L	setMinimumTokenBalanceForDividends	Public		onlyOwner
L	addLiquidity	Private		
L	forceSwapAndSendDividends	Public		onlyOwner
L	swapAndSendDividends	Private		
L	multiSend	Public		onlyOwner
L	airdropToWallets	External		onlyOwner
PhoenixTokenDividendTracker	Implementation	DividendPayingToken, Ownable		
L		Public		DividendPayingToken
L	decimals	Public		NO!
L	name	Public		NO!
L	symbol	Public		NO!
L	_transfer	Internal		
L	withdrawDividend	Public		NO!
L	isExcludedFromAutoClaim	External		onlyOwner
L	isReinvest	External		onlyOwner
L	setAllowCustomTokens	External		onlyOwner
L	setAllowAutoReinvest	External		onlyOwner

Contract	Type	Bases		
L	excludeFromDividends	External		onlyOwner
L	includeFromDividends	External		onlyOwner
L	setAutoClaim	External		onlyOwner
L	setReinvest	External		onlyOwner
L	setMinimumTokenBalanceForAutoDividends	External		onlyOwner
L	setMinimumTokenBalanceForDividends	External		onlyOwner
L	setDividendsPaused	External		onlyOwner
L	getLastProcessedIndex	External		NO!
L	getNumberOfTokenHolders	External		NO!
L	getAccount	Public		NO!
L	getAccountAtIndex	Public		NO!
L	setBalance	External		onlyOwner
L	process	Public		NO!
L	processAccount	Public		onlyOwner
L	updateUniswapV2Router	Public		onlyOwner
L	updatePayoutToken	Public		onlyOwner
L	getPayoutToken	Public		NO!
L	_reinvestDividendOfUser	Private		
L	_withdrawDividendOfUser	Internal		

Contract	Type	Bases		
IterableMapping	Library			
L	get	Internal 🔒		
L	getIndexOfKey	Internal 🔒		
L	getKeyAtIndex	Internal 🔒		
L	size	Internal 🔒		
L	set	Internal 🔒	🛑	
L	remove	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