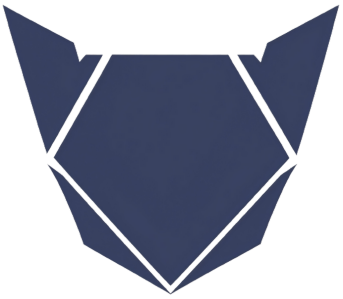


SPHINXSHIELD

Security Assessment

pETHEREUM

Nov 8th, 2023

Evaluation Outcomes

Security Score

Review	Score
Overall Score	73/100
Auditor Score	69/100

Review by Section	Score
Manual Scan Score	30/57
Advance Check Score	8/19

Scoring System

This scoring system is provided to gauge the overall value of the audit. The maximum achievable score is 100, but reaching this score requires the project to meet all assessment requirements.

Our updated passing score is now set at 80 points. If a project fails to achieve at least 80% of the total score, it will result in an automatic failure.

Please refer to our notes and final assessment for more details.

Table of Contents

Summary

This audit report is tailored for **pETHEREUM**, aiming to uncover potential issues and vulnerabilities within the **pETHEREUM** project's source code, along with scrutinizing contract dependencies outside recognized libraries. Our audit comprises a comprehensive investigation involving Static Analysis and Manual Review techniques.

Our audit process places a strong emphasis on the following focal points:

1. Rigorous testing of smart contracts against both commonplace and rare attack vectors.
2. Evaluation of the codebase for alignment with contemporary best practices and industry standards.
3. Ensuring the contract logic is in harmony with the client's specifications and objectives.
4. A comparative analysis of the contract structure and implementation against analogous smart contracts created by industry frontrunners.
5. An exhaustive, line-by-line manual review of the entire codebase by domain experts.

The outcome of this security assessment yielded findings spanning from critical to informational. To uphold robust security standards and align with industry norms, we present the following security-driven recommendations:

1. Elevate general coding practices to optimize source code structure.
2. Implement an all-encompassing suite of unit tests to account for all conceivable use cases.
3. Enhance codebase transparency through increased commenting, particularly in externally verifiable contracts.
4. Improve clarity regarding privileged activities upon the protocol's transition to a live state.

Overview

Project Summary

Project Name	pETHEREUM on PULSECHAIN
Blockchain	Pulse Chain
Language	Solidity
Codebase	https://scan.pulsechain.com/token/0x6Cbd8D593475413F1fb8015b8E5e918c1dd495C4
Commit	aab9478c4f93d3fdc04aa833a8eb5d152928dd9867d34929ef6652c78701fa26

Audit Summary

Delivery Date	Nov 8th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	PETHEREUM.sol

Vulnerability Summary

37
Total Findings

0	0	0	37	0
Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
High	10	0	0	10	0
Medium	1	0	0	1	0
Low	3	0	0	3	0
Informational	23	0	0	23	0
Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
ERS	PETHEREUM.sol	0xbb68628283be05d9d714000ab396cf4c21efc459ed84804d4f57600e9767e17e

Understandings

pETHEREUM (ERS) is an Ethereum-based token deployed on the PULSECHAIN.

Token Information

- Token Name: pETHEREUM on PULSECHAIN
- Symbol: ERS
- Decimals: 9
- Total Supply: 100,000,000 ERS

Tax Distribution

Transactions with pETHEREUM incur a total fee of 10%, which is distributed among different components:

- Liquidity Fee (1%): Collected for providing liquidity to the token. Set by the owner.
- Reflection Fee (7%): A portion goes to holders as reflections.
- Marketing Fee (2%): Allocated for marketing efforts.

Fee Management

The contract allows the owner to manage various fees, including liquidity, reflection, and marketing fees. This flexibility enables adjustments to the fee structure.

Tax Exemption

The contract provides the owner with the ability to exempt specific addresses from fees. This feature allows whitelisting particular wallets or contracts, exempting them from transaction fees.

Ownership and Authorization

The contract follows an ownership model where certain functions are restricted to the owner. The owner can authorize specific addresses to access privileged functions, ensuring control over critical aspects of the contract.

Transaction Limits

The contract enforces transaction limits to prevent excessive token movement, ensuring that users do not exceed the defined limits during transactions.

Swap Mechanism

pETHEREUM employs a swap mechanism to manage liquidity. When a set threshold of tokens is reached, a portion of the contract's balance is swapped to WPLS tokens via the PancakeSwap Router. This swap action may temporarily affect the token's price. The remaining balance is then supplied to the pETHEREUM-WPLS liquidity pool.

Open Trading

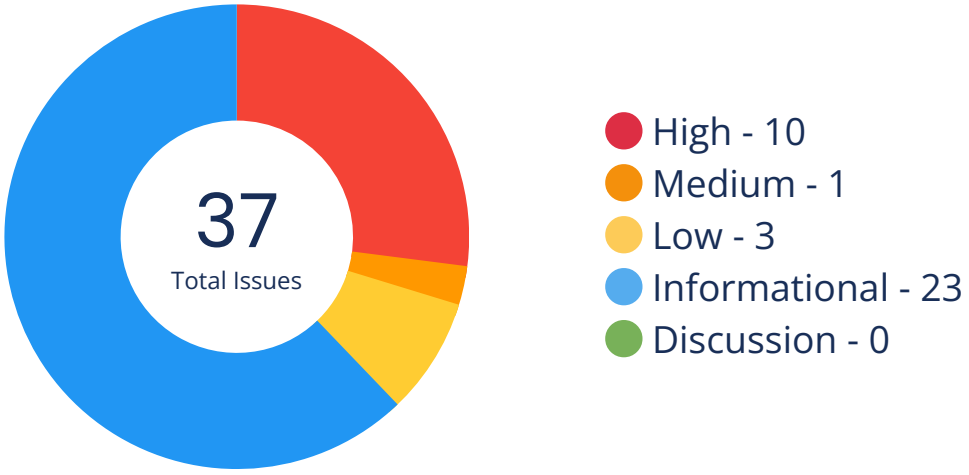
Trading can be restricted based on conditions defined by the owner. This feature ensures that trading remains closed until specific requirements are met, providing additional control over the token's market activity.

Additional Functionality

The contract includes various additional functions, such as airdrop capability, the ability to clear stuck balances in ETH, and more. These functions contribute to the overall functionality and flexibility of the pETHEREUM contract.

This contract embodies pETHEREUM's vision of providing automatic ETHEREUM rewards to its holders. The contract includes essential functions for token transfers, fee management, liquidity, and dividend distribution. The AutoLiquify event signals the automatic liquidity provision during certain conditions.

Findings



Location	Title	Scope	Severity	Status
PETHEREUM.sol:311	Integer Overflow/Underflow	DividendDistributor	High	Aknowledged
PETHEREUM.sol:312	Integer Overflow/Underflow	DividendDistributor	High	Aknowledged
PETHEREUM.sol:485	Integer Overflow/Underflow	PETHEREUM	High	Aknowledged
PETHEREUM.sol:554	Integer Overflow/Underflow	PETHEREUM	High	Aknowledged
PETHEREUM.sol:254	Reentrancy	DividendDistributor	High	Aknowledged
PETHEREUM.sol:270	Reentrancy	DividendDistributor	High	Aknowledged
PETHEREUM.sol:290	Reentrancy	DividendDistributor	High	Aknowledged
PETHEREUM.sol:334	Reentrancy	DividendDistributor	High	Aknowledged

Location	Title	Scope	Severity	Status
PETHEREUM.sol:619	Reentrancy	PETHEREUM	High	Aknowledged
PETHEREUM.sol:687	Reentrancy	PETHEREUM	High	Aknowledged
PETHEREUM.sol:327,597	Unchecked Call Return Value	DividendDistributor	Medium	Aknowledged
PETHEREUM.sol:327,554	Use Safer Functions	DividendDistributor	Low	Aknowledged
PETHEREUM.sol:231,406	Uninitialized Variables	DividendDistributor	Low	Aknowledged
PETHEREUM.sol:679	Division Before Multiplication	PETHEREUM	Low	Aknowledged
PETHEREUM.sol:412	Prefer uint256	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:255,322,339,342,350,359,423,439,483	Cache State Variables that are Read Multiple Times within A Function	DividendDistributor	Informational	Aknowledged
PETHEREUM.sol:227,317,359,360,378,379,381,416,478,485,494,554,694	Use SafeMath When Compiler Version below 0.8.0	DividendDistributor	Informational	Aknowledged
PETHEREUM.sol:37,259	Use != 0 Instead of > 0 for Unsigned Integer Comparison	DividendDistributor	Informational	Aknowledged
PETHEREUM.sol:3	Floating Pragma	Global	Informational	Aknowledged
PETHEREUM.sol:28,485,493,691	Long String in revert/require	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:553,578	Get Contract Balance of ETH in Assembly	PETHEREUM	Informational	Aknowledged

Location	Title	Scope	Severity	Status
PETHEREUM.sol:243	Use Assembly to Check Zero Address	DividendDistributor	Informational	Aknowledged
PETHEREUM.sol:410	Too Many Digits	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:554	Prefer .call() To send()/transfer()	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:3,45,61,183,365	Recommend to Follow Code Layout Conventions	IBEP20	Informational	Aknowledged
PETHEREUM.sol:65,87,94,115,254,454,460,464,468,619,629,633,637,650,687	No Check of Address Params with Zero Address	Auth	Informational	Aknowledged
PETHEREUM.sol:224,368,369,370,371,372,378	Variables Should Be Constants	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:27,38	Use Shift Operation Instead of Mul/Div	SafeMath	Informational	Aknowledged
PETHEREUM.sol:693,699	Use ++i/--i Instead of i++/i--	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:266	Continuous State Variable Write	DividendDistributor	Informational	Aknowledged
PETHEREUM.sol:116	Event Should be Emitted When Critical State Variables Change	Auth	Informational	Aknowledged
PETHEREUM.sol:558,562	Function Visibility Can Be External	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:3	Usage of Outdated Compiler	Global	Informational	Aknowledged

Location	Title	Scope	Severity	Status
PETHEREUM.sol:9,17,28,37,74,81,239,485,493,529,620,647,670,691,697	Use CustomError Instead of String	Auth	Informational	Aknowledged
PETHEREUM.sol:239,620,647,670	Lack of Error Message	DividendDistributor	Informational	Aknowledged
PETHEREUM.sol:569	ReentrancyGuard Should Modify External Function	PETHEREUM	Informational	Aknowledged
PETHEREUM.sol:186,212,403,404,409	Variables Can Be Declared as Immutable	DividendDistributor	Informational	Aknowledged

Code Security - Integer Overflow/Underflow

Title	Severity	Location	Status
Integer Overflow/Underflow	High	PETHEREUM.sol:311	Aknowledged

Description

An overflow/underflow may happen when an arithmetic operation reaches the maximum or minimum size of a type.

Code Security - Integer Overflow/Underflow

Title	Severity	Location	Status
Integer Overflow/Underflow	High	PETHEREUM.sol:312	Aknowledged

Description

An overflow/underflow may happen when an arithmetic operation reaches the maximum or minimum size of a type.

Code Security - Integer Overflow/Underflow

Title	Severity	Location	Status
Integer Overflow/Underflow	High	PETHEREUM.sol:485	Aknowledged

Description

An overflow/underflow may happen when an arithmetic operation reaches the maximum or minimum size of a type.

Code Security - Integer Overflow/Underflow

Title	Severity	Location	Status
Integer Overflow/Underflow	High	PETHEREUM.sol:554	Aknowledged

Description

An overflow/underflow may happen when an arithmetic operation reaches the maximum or minimum size of a type.

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	High	PETHEREUM.sol:254	Aknowledged

Description

As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	High	PETHEREUM.sol:270	Aknowledged

Description

As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	High	PETHEREUM.sol:290	Aknowledged

Description

As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	High	PETHEREUM.sol:334	Aknowledged

Description

The input parameter of As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.address type in the function does not use the zero address for verification.

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	High	PETHEREUM.sol:619	Aknowledged

Description

The input parameter of As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.address type in the function does not use the zero address for verification.

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	High	PETHEREUM.sol:687	Aknowledged

Description

The input parameter of As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.address type in the function does not use the zero address for verification.

Code Security - Unchecked Call Return Value

Title	Severity	Location	Status
Unchecked Call Return Value	Medium	PETHEREUM.sol:327,597	Aknowledged

Description

The return value of low level calls and external calls (transfer, transferFrom and approve) should be verified since low level calls may fail and these three external function calls may only return false but not cause execution reverted once fail. If not properly handled, it might incur asset losses to users and the project party.

Code Security - Use Safer Functions

Title	Severity	Location	Status
Use Safer Functions	Low	PETHEREUM.sol:327,554	Aknowledged

Description

When calling the transfer, transferFrom, and approve functions in the ERC20 contract, there are some contracts that are not fully implemented in accordance with the ERC20 standard. In order to more comprehensively judge whether the call result meets expectations or to be compatible with different ERC20 contracts, it is recommended to use the safeTransfer, safeTransferFrom, safeApprove function to call.

Code Security - Uninitialized Variables

Title	Severity	Location	Status
Uninitialized Variables	Low	PETHEREUM.sol:231,406	Aknowledged

Description

Variables that are not initialized after definition are used in the contract.

Code Security - Division Before Multiplication

Title	Severity	Location	Status
Division Before Multiplication	Low	PETHEREUM.sol:679	Aknowledged

Description

Solidity operates only with integers. Thus, if the division is done before the multiplication, the rounding errors can increase dramatically.

Optimization Suggestion - Prefer uint256

Title	Severity	Location	Status
Prefer uint256	Informational	PETHEREUM.sol:412	Aknowledged

Description

It is recommended to use uint256/int256 types to avoid gas overhead caused by 32 bytes padding.

Optimization Suggestion - Cache State Variables that are Read Multiple Times within A Function

Title	Severity	Location	Status
Cache State Variables that are Read Multiple Times within A Function	Informational	PETHEREUM.sol:255,322,339,342,350,359,423,439,483	Aknowledged

Description

When a state variable is read multiple times in a function, using a local variable to cache the state variable can avoid frequently reading data from storage, thereby saving gas.

Optimization Suggestion - Use SafeMath When Compiler Version below 0.8.0

Title	Severity	Location	Status
Use SafeMath When Compiler Version below 0.8.0	Informational	PETHEREUM.sol:227,317,359,360,378,379,381,416,478,485,494,554,694	Aknowledged

Description

Contracts using compiler versions below 0.8.0 are recommended to use the SafeMath library to prevent overflow of arithmetic operation.

Optimization Suggestion - Use != 0 Instead of > 0 for Unsigned Integer Comparison

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	Informational	PETHEREUM.sol:37,259	Aknowledged

Description

For unsigned integers, use !=0 for comparison, which consumes less gas than >0. When compiler optimization is turned off, about 3 gas can be saved. When compiler optimization is turned on, no gas can be saved.

Optimization Suggestion - Floating Pragma

Title	Severity	Location	Status
Floating Pragma	Informational	PETHEREUM.sol:3	Aknowledged

Description

Contracts should be deployed with fixed compiler version which has been tested thoroughly or make sure to lock the contract compiler version in the project configuration. Locked compiler version ensures that contracts will not be compiled by untested compiler version.

Optimization Suggestion - Long String in revert/require

Title	Severity	Location	Status
Long String in revert/require	Informational	PETHEREUM.sol:28,485,493,691	Aknowledged

Description

If the string parameter in the revert/require function exceeds 32 bytes, more gas will be consumed.

Optimization Suggestion - Get Contract Balance of ETH in Assembly

Title	Severity	Location	Status
Get Contract Balance of ETH in Assembly	Informational	PETHEREUM.sol:553,578	Aknowledged

Description

Using the selfbalance and balance opcodes to get the ETH balance of the contract in assembly saves gas compared to getting the ETH balance through address(this).balance and xx.balance. When compiler optimization is turned off, about 210-250 gas can be saved, and when compiler optimization is turned on, about 50-100 gas can be saved.

Optimization Suggestion - Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	Informational	PETHEREUM.sol:243	Aknowledged

Description

Using assembly to check zero address can save gas. About 18 gas can be saved in each call.

Optimization Suggestion - Too Many Digits

Title	Severity	Location	Status
Too Many Digits	Informational	PETHEREUM.sol:410	Aknowledged

Description

The number is too long, and it is easy to make mistakes when modifying and maintaining.

Optimization Suggestion - Prefer .call() To send()/transfer()

Title	Severity	Location	Status
Prefer .call() To send()/transfer()	Informational	PETHEREUM.sol:554	Aknowledged

Description

The send or transfer function has a limit of 2300 gas.

Optimization Suggestion - Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	Informational	PETHEREUM.sol:3,45,61,183,365	Aknowledged

Description

In the solidity document (https://docs.soliditylang.org/en/v0.8.17/style-guide.html), there are the following conventions for code layout: Layout contract elements in the following order: 1. Pragma statements, 2. Import statements, 3. Interfaces, 4. Libraries, 5. Contracts. Inside each contract, library or interface, use the following order: 1. Type declarations, 2. State variables, 3. Events, 4. Modifiers, 5. Functions. Functions should be grouped according to their visibility and ordered: 1. constructor, 2. receive function (if exists), 3. fallback function (if exists), 4. external, 5. public, 6. internal, 7. private.

Optimization Suggestion - No Check of Address Params with Zero Address

Title	Severity	Location	Status
No Check of Address Params with Zero Address	Informational	PETHEREUM.sol:65,87,94,115,254,454,460,464,468,619,629,633,637,650,687	Aknowledged

Description

The input parameter of the address type in the function does not use the zero address for verification.

Optimization Suggestion - Variables Should Be Constants

Title	Severity	Location	Status
Variables Should Be Constants	Informational	PETHEREUM.sol:224,368,369,370,371,372,378	Aknowledged

Description

There are unchanging state variables in the contract, and putting unchanging state variables in storage will waste gas.

Optimization Suggestion - Use Shift Operation Instead of Mul/Div

Title	Severity	Location	Status
Use Shift Operation Instead of Mul/Div	Informational	PETHEREUM.sol:27,38	Aknowledged

Description

It is recommended to use shift operation instead of direct multiplication and division if possible, because shift operation is more gas-efficient.

Optimization Suggestion - Use ++i/--i Instead of i++/i--

Title	Severity	Location	Status
Use ++i/--i Instead of i++/i--	Informational	PETHEREUM.sol:693,699	Aknowledged

Description

Compared with i++, ++i can save about 5 gas per use. Compared with i--, --i can save about 3 gas per use in for loop.

Optimization Suggestion - Continuous State Variable Write

Title	Severity	Location	Status
Continuous State Variable Write	Informational	PETHEREUM.sol:266	Aknowledged

Description

When there are multiple continuous write operations on a state variable, the intermediate write operations are redundant and will cost more gas.

Optimization Suggestion - Event Should be Emitted When Critical State Variables Change

Title	Severity	Location	Status
Event Should be Emitted When Critical State Variables Change	Informational	PETHEREUM.sol:116	Aknowledged

Description

When some critical variables in the contract, such as owner and balance change, an event should be emitted so that the changes of these variables can be tracked off-chain.

Optimization Suggestion - Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	Informational	PETHEREUM.sol:558,562	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Usage of Outdated Compiler

Title	Severity	Location	Status
Usage of Outdated Compiler	Informational	PETHEREUM.sol:3	Aknowledged

Description

The outdated compiler version (below solidity 0.8.0) is used in the contract, and there may be some security vulnerabilities.

Optimization Suggestion - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	Informational	PETHEREUM.sol:9,17,28,37,74,81,239,485,493,529,620,647,670,691,697	Aknowledged

Description

When using require or revert, CustomError is more gas efficient than string description, as the error message described using CustomError is only compiled into four bytes. Especially when string exceeds 32 bytes, more gas will be consumed. Generally, around 250-270 gas can be saved for one CustomError replacement when compiler optimization is turned off, 60-80 gas can be saved even if compiler optimization is turned on.

Optimization Suggestion - Lack of Error Message

Title	Severity	Location	Status
Lack of Error Message	Informational	PETHEREUM.sol:239,620,647,670	Aknowledged

Description

Use empty string as parameter while invoking function revert or require.

Optimization Suggestion - ReentrancyGuard Should Modify External Function

Title	Severity	Location	Status
ReentrancyGuard Should Modify External Function	Informational	PETHEREUM.sol:569	Aknowledged

Description

The reentrancy guard modifier should modify the external function, because reentrancy vulnerabilities often occur in external calls.

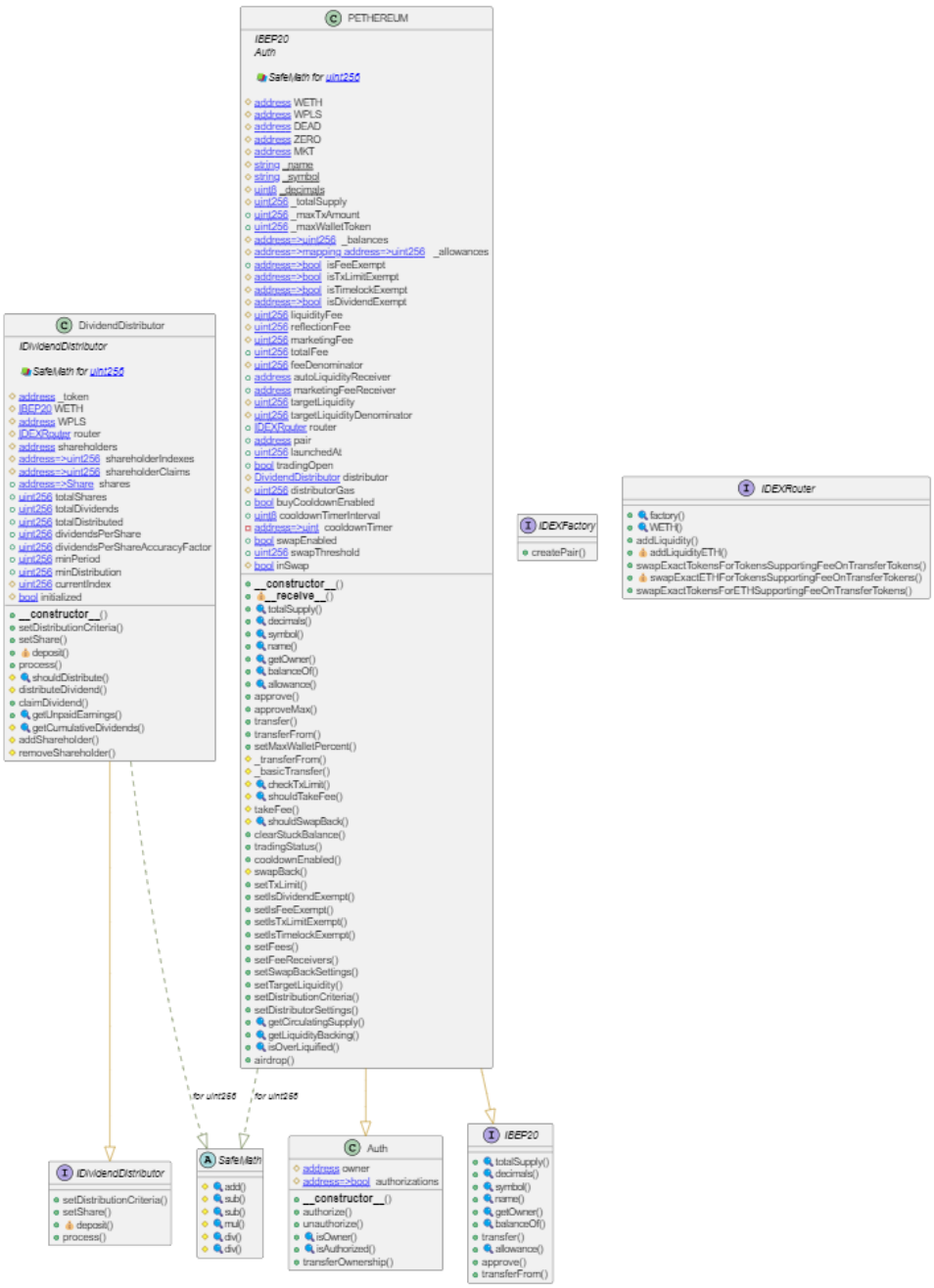
Optimization Suggestion - Variables Can Be Declared as Immutable

Title	Severity	Location	Status
Variables Can Be Declared as Immutable	Informational	PETHEREUM.sol:186,212,403,404,409	Aknowledged

Description

The solidity compiler of version 0.6.5 introduces immutable to modify state variables that are only modified in the constructor. Using immutable can save gas.

PlantUML


PETHEREUM
IBEP20
Auth
SafeMath for uint256
address WETH
address WPLS
address DEAD
address ZERO
address MKT
string _name
string _symbol
uint8 _decimals
uint256 _totalSupply
uint256 _maxTxAmount
uint256 _maxWalletToken
address=>uint256 _balances
address=>mapping address=>uint256 _allowances
address=>bool isFeeExempt
address=>bool isTxLimitExempt
address=>bool isTimelockExempt
address=>bool isDividendExempt
uint256 liquidityFee
uint256 reflectionFee
uint256 marketingFee
uint256 totalFee
uint256 feeDenominator
address autoLiquidityReceiver
address marketingFeeReceiver
uint256 targetLiquidity
uint256 targetLiquidityDenominator
IDEXRouter router
address pair
uint256 launchedAt
bool tradingOpen
DividendDistributor distributor
uint256 distributorGas
bool buyCooldownEnabled
uint8 cooldownTimerInterval
address=>uint cooldownTimer
bool swapEnabled
uint256 swapThreshold
bool inSwap
__constructor__()
__receive__()
totalSupply()
decimals()
symbol()
name()
getOwner()
balanceOf()
allowance()
approve()
approveMax()
transfer()
transferFrom()
setMaxWalletPercent()
_transferFrom()
_basicTransfer()
checkTxLimit()
shouldTakeFee()
takeFee()
shouldSwapBack()
clearStuckBalance()
tradingStatus()
cooldownEnabled()
swapBack()
setTxLimit()
setIsDividendExempt()
setIsFeeExempt()
setIsTxLimitExempt()
setIsTimelockExempt()
setFees()
setFeeReceivers()
setSwapBackSettings()
setTargetLiquidity()
setDistributionCriteria()
setDistributorSettings()
getCirculatingSupply()
getLiquidityBacking()
isOverLiquified()
airdrop()
DividendDistributor
IDividendDistributor
SafeMath for uint256
address _token
IBEP20 WETH
address WPLS
IDEXRouter router
address shareholders
address=>uint256 shareholderIndexes
address=>uint256 shareholderClaims
address=>Share shares
uint256 totalShares
uint256 totalDividends
uint256 totalDistributed
uint256 dividendsPerShare
uint256 dividendsPerShareAccuracyFactor
uint256 minPeriod
uint256 minDistribution
uint256 currentIndex
bool initialized
__constructor__()
setDistributionCriteria()
setShare()
deposit()
process()
shouldDistribute()
distributeDividend()
claimDividend()
getUnpaidEarnings()
getCumulativeDividends()
addShareholder()
removeShareholder()
IDEXFactory
createPair()
IDEXRouter
factory()
WETH()
addLiquidity()
addLiquidityETH()
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
for uint256
for uint256
IDividendDistributor
setDistributionCriteria()
setShare()
deposit()
process()
SafeMath
add()
sub()
sub()
mul()
div()
div()
Auth
address owner
address=>bool authorizations
__constructor__()
authorize()
unauthorize()
isOwner()
isAuthorized()
transferOwnership()
IBEP20
totalSupply()
decimals()
symbol()
name()
getOwner()
balanceOf()
transfer()
allowance()
approve()
transferFrom()

Appendix

Finding Categories

Security and Best Practices

1. Integer Overflow/Underflow: Smart contracts must implement safeguards against integer overflow and underflow to prevent unexpected behavior and vulnerabilities.
2. Reentrancy: Mitigate reentrancy attacks by ensuring that external calls are made after state changes, minimizing the potential for malicious interactions.
3. Unchecked Call Return Value: Verify return values of external calls to prevent unexpected behavior or vulnerabilities due to unchecked return data.
4. Use Safer Functions: Utilize functions known for their secure design to mitigate potential security vulnerabilities. Review functions for enhanced security.
5. Uninitialized Variables: Always initialize variables before use to prevent unpredictable behavior and potential vulnerabilities.
6. Division Before Multiplication: Execute division operations before multiplication to ensure accurate mathematical computations and avoid unexpected results.
7. Prefer uint256: Emphasize the use of uint256 over other data types to maintain consistency and enhance contract security.
8. Cache State Variables that are Read Multiple Times within A Function: Optimize gas efficiency by caching state variables that are read multiple times within a function.
9. Use SafeMath When Compiler Version below 0.8.0: Implement SafeMath library for arithmetic operations when using Solidity versions below 0.8.0 to prevent overflow and underflow vulnerabilities.
10. Use != 0 Instead of > 0 for Unsigned Integer Comparison: Prefer using != 0 for unsigned integer comparisons to enhance code clarity and avoid potential issues.
11. Floating Pragma: Ensure that your Solidity pragma remains consistent for added contract security.
12. Long String in revert/require: Long revert or require strings can increase gas usage and should be optimized for gas efficiency.
13. Get Contract Balance of ETH in Assembly: Optimize gas usage by using assembly to retrieve the contract's ETH balance.
14. Use Assembly to Check Zero Address: Employ optimized assembly checks to verify zero addresses efficiently.
15. Too Many Digits: Be cautious of using an excessive number of digits, as it can impact gas costs and contract efficiency.
16. Prefer .call() To send()/transfer(): Employ .call() instead of send()/transfer() for external contract calls to minimize security risks.
17. Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability and maintainability.
18. No Check of Address Params with Zero Address: Verification of address parameters should include checks to ensure that the address is not the zero address.
19. Variables Should Be Constants: Declare variables as constants when their values should not be modified to enhance security and readability.
20. Use Shift Operation Instead of Mul/Div: Utilize shift operations instead of multiplication/division for better gas efficiency.
21. Use ++i/--i Instead of i++/i--: Prefer using pre-increment/pre-decrement (++i/--i) for improved gas efficiency.
22. Continuous State Variable Write: Be cautious of continuous state variable writes, as they can impact gas costs and contract efficiency.
23. Event Should be Emitted When Critical State Variables Change: Emit events when critical state variables change to provide transparency and facilitate external monitoring.

1. Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
2. Usage of Outdated Compiler: Use the latest compiler version to leverage security improvements and optimizations.
3. Use CustomError Instead of String: Opt for custom error codes instead of string error messages for more efficient contract operation.
4. Lack of Error Message: Include informative error messages to assist developers and users in understanding and resolving issues.
5. ReentrancyGuard Should Modify External Function: Ensure that ReentrancyGuard is appropriately implemented, modifying external functions to prevent reentrancy attacks.
6. Variables Can Be Declared as Immutable: Declare variables as immutable when their values do not change after initialization to enhance security and readability.

KECCAK256 or SHA256 Checksum Verification

Checksum verification is a critical component of smart contract development. It ensures the integrity of contract deployment and code execution by confirming that the bytecode being executed matches the intended source code. The following details the KECCAK256 and SHA256 checksum verification process.

KECCAK256 Checksum Verification:

- Checksum Definition: KECCAK256 is a cryptographic hashing function used in Ethereum to create a checksum of the contract bytecode. It is part of the Ethereum Name Service (ENS) standard.
- Use Cases: KECCAK256 checksums are used in ENS for verification of Ethereum addresses. They help prevent unintended transfers due to typos or errors.
- Checksum Process: The KECCAK256 checksum is created by taking the SHA3 hash of the lowercase hexadecimal Ethereum address, and then converting it to the corresponding checksum address by replacing characters with uppercase letters.

SHA256 Checksum Verification:

- Checksum Definition: SHA256 is a widely used cryptographic hash function, often employed to verify the integrity of data and contracts.
- Use Cases: SHA256 checksums are widely used in software development, including the verification of software downloads and smart contracts.
- Checksum Process: The SHA256 checksum is generated by applying the SHA256 hashing algorithm to the content of the contract. This results in a fixed-length hexadecimal value that is compared to the expected value to verify the contract's integrity.

Importance of Checksum Verification:

- Checksum verification ensures that smart contracts are executed as intended, preventing tampering and security vulnerabilities.
- It is a security best practice to verify that the deployed bytecode matches the intended source code, reducing the risk of unexpected behavior.

Best Practices:

- Always use checksum verification in situations where it is essential to verify Ethereum addresses or contract integrity.
- Implement checksum verification to ensure that contract deployment and interactions occur as intended.
- Verify the validity of contract deployments and the integrity of the code during development and deployment phases.

Website Scan

https://pethereum.finance/

Network Security

High | 0 Attentions

Application Security

High | 6 Attentions

DNS Security

High | 6 Attentions

Network Security

9 Passed **0 Attention**

FTP Service Anonymous LOGIN	NO
VNC Service Accesible	NO
RDP Service Accesible	NO
LDAP Service Accesible	NO
PPTP Service Accesible	NO
RSYNC Service Accesible	NO
SSH Weak Cipher	NO
SSH Support Weak MAC	NO
CVE on the Related Service	NO

Application Security

5 Passed | 6 Attention

Missing X-Frame-Options Header	YES
Missing HSTS header	YES
Missing X-Content-Type-Options Header	YES
Missing Content Security Policy (CSP)	YES
HTTP Access Allowed	NO
Self-Signed Certificate	NO
Wrong Host Certificate	NO
Expired Certificate	NO
SSL/TLS Supports Weak Cipher	YES
Support SSL Protocols	NO
Support TLS Weak Version	YES

DNS Health

4 Passed	6 Attention

Missing SPF Record	YES
Missing DMARC Record	YES
Missing DKIM Record	YES
Ineffective SPF Record	YES
SPF Record Contains a Softfail Without DMARC	YES
Name Servers Versions Exposed	NO
Allow Recursive Queries	YES
CNAME in NS Records	NO
MX Records IPs are Private	NO
MX Records has Invalid Chars	NO

Social Media Checks

2 Passed | 8 Failed

X (Twitter)		PASS
Facebook		FAIL
Instagram		FAIL
TikTok		FAIL
YouTube		FAIL
Twich		FAIL
Telegram		PASS
Discord		FAIL
Medium		FAIL
Others		FAIL

Recommendation

To enhance project credibility and outreach, we suggest having a minimum of three active social media channels and a fully functional website.

Social Media Information Notes

Unspecified Auditor Notes

Notes from the Project Owner

Fundamental Health

KYC Status

SphinxShield KYC	NO
3rd Party KYC	NO

Project Maturity Metrics

Minimally Developed	**LOW**
Token Launch Date	**2023.10.31 18:00 (UTC)**
Token Market Cap (estimate)	**$819.28**
Token/Project Age	**13 Days**

Recommendation

We strongly recommend that the project undergo the Know Your Customer (KYC) verification process with SphinxShield to enhance transparency and build trust within the crypto community. Furthermore, we encourage the project team to reach out to us promptly to rectify any inaccuracies or discrepancies in the provided information to ensure the accuracy and reliability of their project data.

Coin Tracker Analytics

Status

CoinMarketCap	NO
CoinGecko	NO
Others	NO

Recommendation

We highly recommend that the project consider integrating with multiple coin tracking platforms to expand its visibility within the cryptocurrency ecosystem. In particular, joining prominent platforms such as CoinMarketCap and CoinGecko can significantly benefit the project by increasing its reach and credibility.

CEX Holding Analytics

Status

Not available on any centralized cryptocurrency exchanges (CEX).

Recommendation

To increase your project's visibility and liquidity, we recommend pursuing listings on centralized cryptocurrency exchanges. Here's a recommendation you can use:

We strongly advise the project team to actively pursue listings on reputable centralized cryptocurrency exchanges. Being listed on these platforms can offer numerous advantages, such as increased liquidity, exposure to a broader range of traders, and enhanced credibility within the crypto community.

To facilitate this process, we recommend the following steps:

1. Research and Identify Suitable Exchanges: Conduct thorough research to identify centralized exchanges that align with your project's goals and target audience. Consider factors such as trading volume, reputation, geographical reach, and compliance with regulatory requirements.
2. Meet Compliance Requirements: Ensure that your project is compliant with all necessary legal and regulatory requirements for listing on these exchanges. This may include Know Your Customer (KYC) verification, security audits, and legal documentation.
3. Prepare a Comprehensive Listing Proposal: Create a detailed and persuasive listing proposal for each exchange you intend to approach. This proposal should highlight the unique features and benefits of your project, as well as your commitment to compliance and security.
4. Engage in Communication: Establish open lines of communication with the exchange's listing team. Be prepared to address their questions, provide requested documentation, and work closely with their team to facilitate the listing process.
5. Marketing and Community Engagement: Promote your project within the exchange's community and among your own supporters to increase visibility and trading activity upon listing.
6. Maintain Transparency: Maintain transparency and provide regular updates to your community and potential investors about the progress of listing efforts.
7. Be Patient and Persistent: Listing processes on centralized exchanges can sometimes be lengthy. Be patient and persistent in your efforts, and consider seeking the assistance of experts or advisors with experience in exchange listings if necessary.
8.

Remember that listing on centralized exchanges can significantly impact your project's growth and market accessibility. By following these steps and maintaining a professional, compliant, and communicative approach, you can increase your chances of successfully getting listed on centralized exchanges.

Disclaimer

SphinxShield, its agents, and associates provide the information and content contained within this audit report and materials (collectively referred to as "the Services") for informational and security assessment purposes only. The Services are provided "as is" without any warranty or representation of any kind, either express or implied. SphinxShield, its agents, and associates make no warranty or undertaking, and do not represent that the Services will:

- Meet customer's specific requirements.
- Achieve any intended results.
- Be compatible or work with any other software, applications, systems, or services.
- Operate without interruption.
- Meet any performance or reliability standards.
- Be error-free or that any errors or defects can or will be corrected.

In addition, SphinxShield, its agents, and associates make no representation or warranty of any kind, express or implied, as to the accuracy, reliability, or currency of any information or content provided through the Services. SphinxShield assumes no liability or responsibility for any:

- Errors, mistakes, or inaccuracies of content and materials.
- Loss or damage of any kind incurred as a result of the use of any content.
- Personal injury or property damage, of any nature whatsoever, resulting from customer's access to or use of the Services, assessment report, or other materials.

It is imperative to recognize that the Services, including any associated assessment reports or materials, should not be considered or relied upon as any form of financial, tax, legal, regulatory, or other advice.

SphinxShield provides the Services solely to the customer and for the purposes specifically identified in this agreement. The Services, assessment reports, and accompanying materials may not be relied upon by any other person or for any purpose not explicitly stated in this agreement. Copies of these materials may not be delivered to any other person without SphinxShield's prior written consent in each instance.

Furthermore, no third party or anyone acting on behalf of a third party shall be considered a third-party beneficiary of the Services, assessment reports, and any accompanying materials. No such third party shall have any rights of contribution against SphinxShield with respect to the Services, assessment reports, and any accompanying materials.

The representations and warranties of SphinxShield contained in this agreement are solely for the benefit of the customer. Accordingly, no third party or anyone acting on behalf of a third party shall be considered a third-party beneficiary of such representations and warranties. No such third party shall have any rights of contribution against SphinxShield with respect to such representations or warranties or any matter subject to or resulting in indemnification under this agreement or otherwise.

About

SphinxShield, established in 2023, is a cybersecurity and auditing firm dedicated to fortifying blockchain and cryptocurrency security. We specialize in providing comprehensive security audits and solutions, aimed at protecting digital assets and fostering a secure investment environment.

Our accomplished team of experts possesses in-depth expertise in the blockchain space, ensuring our clients receive meticulous code audits, vulnerability assessments, and expert security advice. We employ the latest industry standards and innovative auditing techniques to reveal potential vulnerabilities, guaranteeing the protection of our clients' digital assets against emerging threats.

At SphinxShield, our unwavering mission is to promote transparency, security, and compliance with industry standards, contributing to the growth of blockchain and cryptocurrency projects. As a forward-thinking company, we remain adaptable, staying current with emerging trends and technologies to consistently enhance our services.

SphinxShield is your trusted partner for securing crypto ventures, empowering you to explore the vast potential of blockchain technology with confidence.