Cyberscope

Audit Report

Shiboost

March 2023

Network	ETH
Address	0x1322310635cc6bB6435E70120c563535e3D0CF86
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 4
Findings Breakdown **5**
Analysis **6**
ST - Stops Transactions 7
Description 7
Recommendation 7
OCTD - Transfers Contract's Tokens 8
Description 8
Recommendation 8
Diagnostics **9**
PTRP - Potential Transfer Revert Propagation 10
Description 10
Recommendation 10
PVC - Price Volatility Concern 11
Description 11
Recommendation 11
RSML - Redundant SafeMath Library 12
Description 12
Recommendation 12
L02 - State Variables could be Declared Constant 13
Description 13
Recommendation 13
L04 - Conformance to Solidity Naming Conventions 14
Description 14
Recommendation 15
L07 - Missing Events Arithmetic 16
Description 16
Recommendation 16
L09 - Dead Code Elimination 17
Description 17
Recommendation 17
L16 - Validate Variable Setters 18
Description 18
Recommendation 18
L19 - Stable Compiler Version 19
Description 19

Recommendation 19
L20 - Succeeded Transfer Check 20
Description 20
Recommendation 20
Functions Analysis **21**
Inheritance Graph **28**
Flow Graph **29**
Summary **30**
Disclaimer **31**
About Cyberscope **32**

Review

Contract Name	Shiboost
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://etherscan.io/address/0x1322310635cc6bb6435e70120c563535e3d0cf86
Address	0x1322310635cc6bb6435e70120c563535e3d0cf86
Network	ETH
Symbol	SBT
Decimals	18
Total Supply	10,000,000

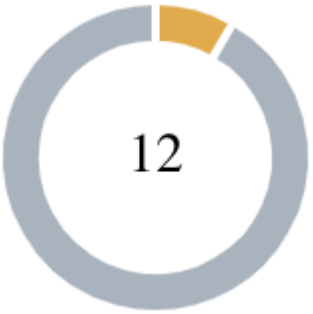
Audit Updates

Initial Audit	30 Mar 2023 https://github.com/cyberscope-io/audits/tree/main/1-shiboost/v1/audit.pdf
Corrected Phase 2	02 Apr 2023 https://github.com/cyberscope-io/audits/tree/main/1-shiboost/v1/audit.pdf
Corrected Phase 3	03 Apr 2023

Source Files

Filename	SHA256
Shiboost.sol	d3d3724925b9876e0e33c6db6fb7c6e30712b63ced848ef0fccfa9a0e80ea412

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Medium
Location	contracts/testingDeploy/shiboost.sol#L618,637
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` and `_walletMax` to zero.

```
if(!isTxLimitExempt[sender] && !isTxLimitExempt[recipient] &&
EnableTransactionLimit) {
    require(amount <= _maxTxAmount, "Transfer amount exceeds the
maxTxAmount.");
}
...
if(checkWalletLimit && !isWalletLimitExempt[recipient]) {
    require(balanceOf(recipient).add(finalAmount) <=
_walletMax,"Amount Exceed From Max Wallet Limit!!");
}
```

Recommendation

The contract could embody a check for not allowing setting the _maxTxAmount less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L649
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueStuckedToken` function.

```
function rescueStuckedToken(address _token, uint _amount) external
onlyOwner {
    IERC20(_token).transfer(msg.sender,_amount);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L16	Validate Variable Setters	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L687,695
Status	Unresolved

Description

The contract sends funds to `marketingWallet` and `DeveloperWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketingWallet).transfer(amountETHMarketing);
payable(DeveloperWallet).transfer(amountETHDeveloper);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L519
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `setNumTokensBeforeSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setNumTokensBeforeSwap(uint256 newLimit) external
onlyOwner() {
    minimumTokensBeforeSwap = newLimit;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases the gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L329,330,331,357
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
string private _name = "Shiboost"
string private _symbol = "SBT"
uint8 private _decimals = 18
uint256 public feeUnits = 100
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L156,157,173,192,334,340,349,350,351,353,354,355,359,360,366,367,377,503,535,574,582,649,755
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
address payable public DeveloperWallet =
payable(0x8754759f47Ea821E66801e9C654B5872d0F09e71)
mapping (address => uint256) _balances
uint256 public _buyLiquidityFee = 1
uint256 public _buyMarketingFee = 2
uint256 public _buyDeveloperFee = 3
uint256 public _sellLiquidityFee = 1
uint256 public _sellMarketingFee = 2
uint256 public _sellDeveloperFee = 3
uint256 public _totalTaxIfBuying
uint256 public _totalTaxIfSelling

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L508,516,520,575,583
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_maxTxAmount = maxTxAmount
_walletMax  = newLimit
minimumTokensBeforeSwap = newLimit
_buyLiquidityFee = _Liquidity
_sellLiquidityFee = _Liquidity
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L552
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function transferToAddressETH(address payable recipient, uint256
amount) private {
        recipient.transfer(amount);
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L524,528,532
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
marketingWallet = payable(newAddress)
liquidityReciever = payable(newAddress)
DeveloperWallet = payable(newAddress)
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L13
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L650
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(_token).transfer(msg.sender,_amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		

	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-

	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-

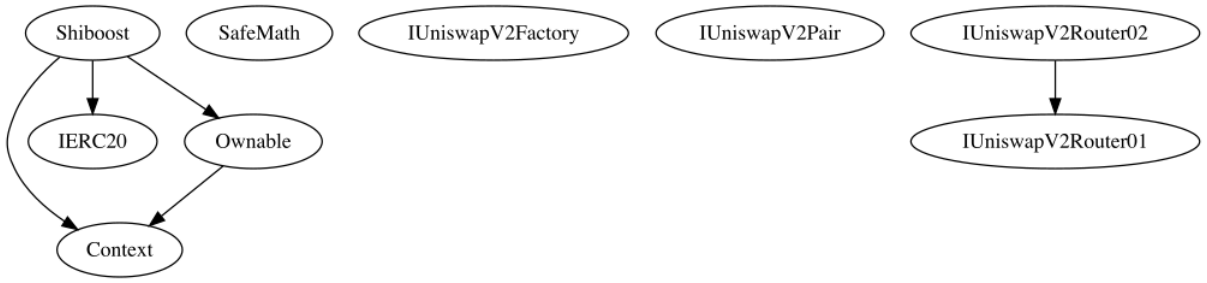
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-

IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Shiboost	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	approve	Public	✓	-
	_approve	Private	✓	
	setMarketPairStatus	Public	✓	onlyOwner
	setIsExcludedFromFee	Public	✓	onlyOwner

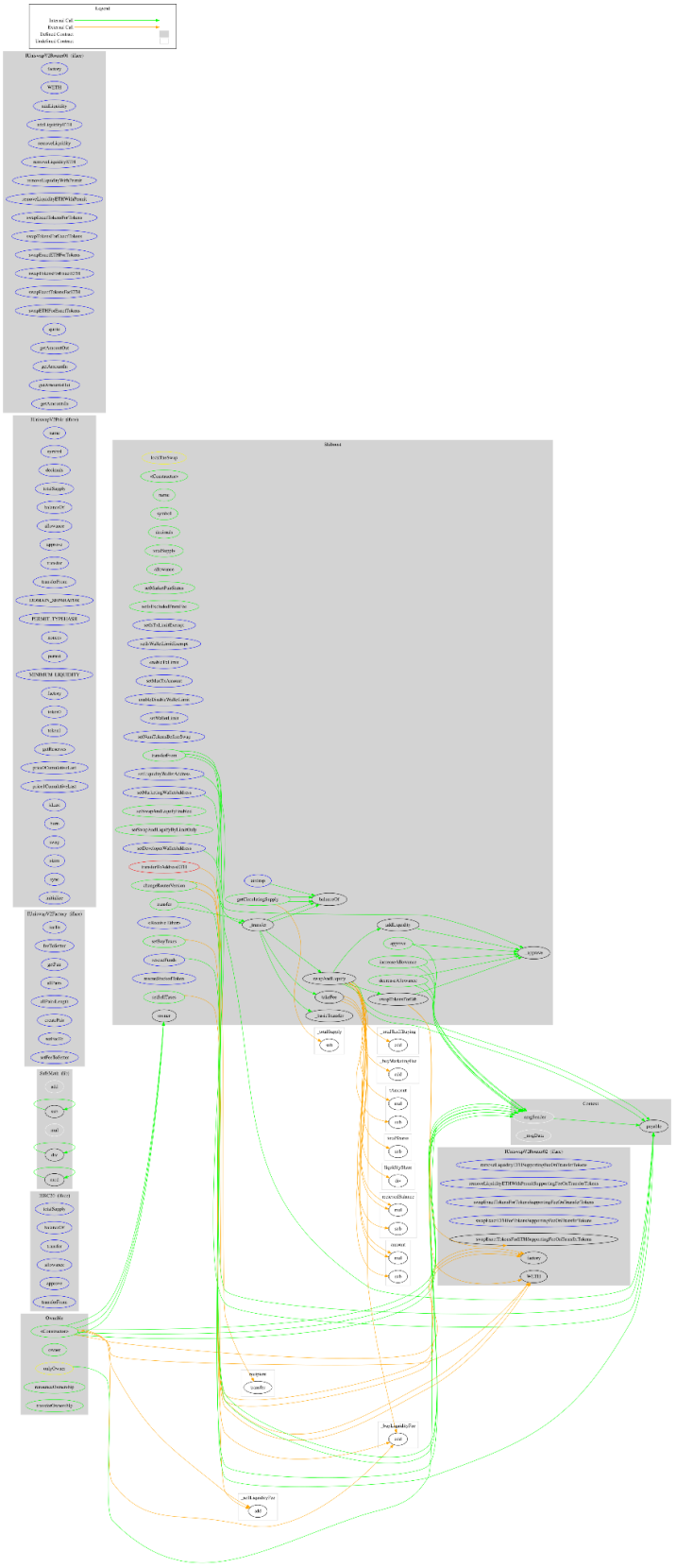
	setIsTxLimitExempt	External	✓	onlyOwner
	setIsWalletLimitExempt	External	✓	onlyOwner
	enableTxLimit	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	enableDisableWalletLimit	External	✓	onlyOwner
	setWalletLimit	External	✓	onlyOwner
	setNumTokensBeforeSwap	External	✓	onlyOwner
	setMarketingWalletAddress	External	✓	onlyOwner
	setLiquidityWalletAddress	External	✓	onlyOwner
	setDeveloperWalletAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setSwapAndLiquifyByLimitOnly	Public	✓	onlyOwner
	getCirculatingSupply	Public		-
	transferToAddressETH	Private	✓	
	changeRouterVersion	Public	✓	onlyOwner
	setBuyTaxes	Public	✓	onlyOwner
	setSellTaxes	Public	✓	onlyOwner
		External	Payable	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Private	✓	
	rescueStuckedToken	External	✓	onlyOwner
	rescueFunds	External	✓	onlyOwner

	_basicTransfer	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	takeFee	Internal	✓	
	airdrop	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Shiboost contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions and draining the contract's tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 25% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

Shiboost

March 2023

Network	ETH
Address	0x06A71bad78318D95Ae8a750e277Da67915FbB615
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Introduction **4**
Roles 4
Owner 4
User 4
Findings Breakdown **5**
Diagnostics **6**
CO - Code Optimization 8
Description 8
Recommendation 8
PSU - Potential Subtraction Underflow 9
Description 9
Recommendation 9
NR - Non-Guaranteed Reward 10
Description 10
Recommendation 11
GO - Gas Optimization 12
Description 12
Recommendation 12
RV - Redundant Variable 13
Description 13
Recommendation 13
RI - Reward Inconsistency 14
Description 14
Recommendation 15
SS - State Synchronization 16
Description 16
Recommendation 16
RSML - Redundant SafeMath Library 17
Description 17
Recommendation 17
RSK - Redundant Storage Keyword 18
Description 18
Recommendation 18
IDI - Immutable Declaration Improvement 19
Description 19

Recommendation 19
L04 - Conformance to Solidity Naming Conventions 20
Description 20
Recommendation 21
L09 - Dead Code Elimination 22
Description 22
Recommendation 23
L17 - Usage of Solidity Assembly 24
Description 24
Recommendation 24
L19 - Stable Compiler Version 25
Description 25
Recommendation 25
L20 - Succeeded Transfer Check 26
Description 26
Recommendation 26
Functions Analysis **27**
Inheritance Graph **32**
Flow Graph **33**
Summary **34**
Disclaimer **35**
About Cyberscope **36**

Review

Contract Name	Staking
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://etherscan.io/address/0x06a71bad78318d95ae8a750e277da67915fbb615
Address	0x06A71bad78318D95Ae8a750e277Da67915FbB615
Network	ETH

Audit Updates

Initial Audit	10 Apr 2023

Source Files

Filename	SHA256
contracts/ShiboostStaking.sol	0c053de45ec38da3c2820b86beee4940530e498e93f8f1cd5d027395b5bbe5c7

Introduction

The ShiboostStaking implements a staking mechanism. Users can deposit tokens to a pool and receive rewards in the form of another token. The contract includes multiple pools, each with its own deposit and withdrawal fees, minimum deposit amount, and harvest interval. The contract also includes a mechanism for locking up rewards and penalizing early withdrawals. The contract owner can add and update pools, start and pause the staking platform, and emergency withdraw tokens from any pool.

Roles

Owner

The owner has authority over the following functions:

- `function initialize()`
- `function add(uint256 _tokenPerBlock, IERC20 _stakedToken, IERC20 _rewardToken, uint16 _depositFeeBP, uint256 _minDeposit, uint256 _harvestInterval, WithdrawFeeInterval[] memory withdrawFeeIntervals)`
- `function set(uint256 _pid, uint256 _tokenPerBlock, uint16 _depositFeeBP, uint256 _minDeposit, uint256 _harvestInterval)`
- `function setWithdrawFeeInterval(uint256 poolId, uint256 index, WithdrawFeeInterval memory _withdrawFeeInterval)`
- `function emergencyAdminWithdraw(uint256 _pid)`
- `function updatePaused(bool _value)`
- `function setLockDeposit(uint256 pid, bool locked)`

User

The user can interact with the following functions:

- `function getWithdrawFeeIntervals(uint256 poolId)`
- `function poolLength()`
- `function getMultiplier(uint256 _from, uint256 _to)`
- `function pendingToken(uint256 _pid, address _user)`
- `function canHarvest(uint256 _pid, address _user)`
- `function depositRewardToken(uint256 poolId, uint256 amount)`
- `function deposit(uint256 _pid, uint256 _amount)`
- `function withdraw(uint256 _pid, uint256 _amount)`
- `function getWithdrawFee(uint256 poolId, uint256 stakedTime)`
- `function emergencyWithdraw(uint256 _pid)`

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CO	Code Optimization	Unresolved
	PSU	Potential Subtraction Underflow	Unresolved
	NR	Non-Guaranteed Reward	Unresolved
	GO	Gas Optimization	Unresolved
	RV	Redundant Variable	Unresolved
	RI	Reward Inconsistency	Unresolved
	SS	State Synchronization	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSK	Redundant Storage Keyword	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L09	Dead Code Elimination	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved

●	L20	Succeeded Transfer Check	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L1002,1106,1115
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `balanceOf` function of an ERC20 token returns an `uint256` number, which means its value will always be greater than or equal to zero. The contract checks if its balance is greater than zero. This operation can be improved by using the `!=` operator.

```
pool.rewardToken.balanceOf(address(this)) > 0
pending > 0 || user.rewardLockedUp > 0
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. As described above, it is recommended to use the `!=` operator.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L1087
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

The `_withdrawFee` must be less than or equal to 1000 for the subtraction to work as expected. The owner has the authority to alter the fees of a certain poolId, and the `setWithdrawFeeInterval` function has no input validation. As a result, the operation may lead to an underflow.

```
uint256 _withdrawFee = getWithdrawFee(_pid, user.depositTimestamp);
uint256 feeAmount = _amount.mul(_withdrawFee).div(1000);
amountToTransfer = _amount.sub(feeAmount);
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

NR - Non-Guaranteed Reward

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L1141
Status	Unresolved

Description

The contract does not guarantee the users will get their rewards. The deposited amounts are transferred to the contract's balance, but the claimable rewards are transferred to the users from the `Reserve` contract. Additionally, the owner has the authority to claim all the reward funds from the `Reserve` contract.

```
function emergencyAdminWithdraw(uint256 _pid) external onlyOwner {
    PoolInfo storage pool = poolInfo[_pid];
    uint256 balanceToWithdraw = pool.rewardToken.balanceOf(address(this));
    require(balanceToWithdraw != 0, "STAKING: Not enough balance to
withdraw!");
    pool.rewardToken.transfer(owner(), balanceToWithdraw);
    rewardReserve.safeTransfer(pool.rewardToken, owner(),
pool.rewardToken.balanceOf(address(rewardReserve)));
    emit AdminEmergencyWithdraw(_pid,
pool.rewardToken.balanceOf(address(this)), pool.accTokenPerShare,
pool.tokenPerBlock, pool.lastRewardBlock);
    delete poolInfo[_pid];
}
```

Recommendation

The contract could guarantee the rewards will be available when a user deposits the amount. The owner is responsible for the `Reserve` contract having all the available funds so that users can claim their reward. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

GO - Gas Optimization

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L875
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `MAXIMUM_HARVEST_INTERVAL` and `harvestInterval` are declared as `uint256` variables. The `MAXIMUM_HARVEST_INTERVAL` is declared as a constant, hence its value cannot change. 14 days is equivalent to `14 * 24 * 60 * 60` seconds. This value can be stored in an `uint32` variable. Storing these variables as `uint256` takes up a lot more storage space, especially in the case of `harvestInterval`, which is a struct property stored in an array. As a result, the contract takes up more storage space than it actually requires, and has increased gas costs for transactions that include this variable.

```
uint256 public constant MAXIMUM_HARVEST_INTERVAL = 14 days;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RV - Redundant Variable

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L874
Status	Unresolved

Description

The `BONUS_MULTIPLIER` is a constant variable with a value of 1. It is only used for multiplication, hence the variable is redundant.

```
uint256 public constant BONUS_MULTIPLIER = 1;
```

Recommendation

The team is advised to remove this variable from the contract as it is redundant.

RI - Reward Inconsistency

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L828
Status	Unresolved

Description

The reward mechanism of the platform is not consistent. The `safeTransfer` function compares the reward amount with the Reserve contract's balance. If the amount is greater, then the user will receive a reward equal to the contract's balance. Since the owner has the authority to claim the balance of the Reserve contract, a user can claim the reward with the possibility of the reward amount the user receives is zero, if the owner previously claimed all the funds. The claiming process will be considered successful though. As a result, the reward mechanism is inconsistent.

```
function safeTransfer(
    IERC20 rewardToken,
    address _to,
    uint256 _amount
) external onlyOwner {
    uint256 tokenBal = rewardToken.balanceOf(address(this));
    if (_amount > tokenBal) {
        rewardToken.transfer(_to, tokenBal);
    } else {
        rewardToken.transfer(_to, _amount);
    }
}
```

Recommendation

The owner is responsible for the Reserve contract having all the available funds so that users can claim their reward. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

SS - State Synchronization

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L995
Status	Unresolved

Description

The contract's state is not being synchronized with what the user sees and the actual values. The `pendingToken` function will return the reward the user is actually entitled to, but in reality, the reward may be much less or even zero. This is a consequence of the RI and NR findings, which are described in detail below.

```
function pendingToken(uint256 _pid, address _user) external view returns
(uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];
    uint256 accTokenPerShare = pool.accTokenPerShare;
    if (
        block.number > pool.lastRewardBlock &&
        pool.stakedAmount != 0 &&
        pool.rewardToken.balanceOf(address(this)) > 0
    ) {
        uint256 multiplier = getMultiplier(pool.lastRewardBlock,
block.number);
        uint256 tokenReward = multiplier.mul(pool.tokenPerBlock);
        accTokenPerShare =
accTokenPerShare.add(tokenReward.mul(1e12).div(pool.stakedAmount));
    }
    return user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt);
}
```

Recommendation

The team is advised to consider all possible outcomes of the reward mechanism and provide realistic data to the users.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, and overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change at https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L996,997,1012,1123
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
PoolInfo storage pool
UserInfo storage user
WithdrawFeeInterval[] storage _withdrawFee
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L899
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
rewardReserve
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L830,831,928,929,930,931,932,933,967,968,969,970,971,984,991,995,1011,1016,1048,1076,1096,1131,1141,1151
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _to
uint256 _amount
uint256 _tokenPerBlock
IERC20 _stakedToken
IERC20 _rewardToken
uint16 _depositFeeBP
uint256 _minDeposit
uint256 _harvestInterval
uint256 _pid
WithdrawFeeInterval memory _withdrawFeeInterval
uint256 _to
uint256 _from
address _user
bool _value
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L393,418,447,480,490,507,517,598,614,623
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }

function functionCall(address target, bytes memory data) internal returns
(bytes memory) {
        return functionCall(target, data, "Address: low-level call
failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L371,546
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            size := extcodesize(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L2,102,112,342,560,660,687,764,826
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L835,837,1145
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
rewardToken.transfer(_to, tokenBal)
rewardToken.transfer(_to, _amount)
pool.rewardToken.transfer(owner(), balanceToWithdraw)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Mintable	Interface	IERC20		
	mint	External	✓	-
	transferOwnership	External	✓	-
SafeMath	Library			
	tryAdd	Internal		

	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	

	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ReentrancyGuard	Implementation			

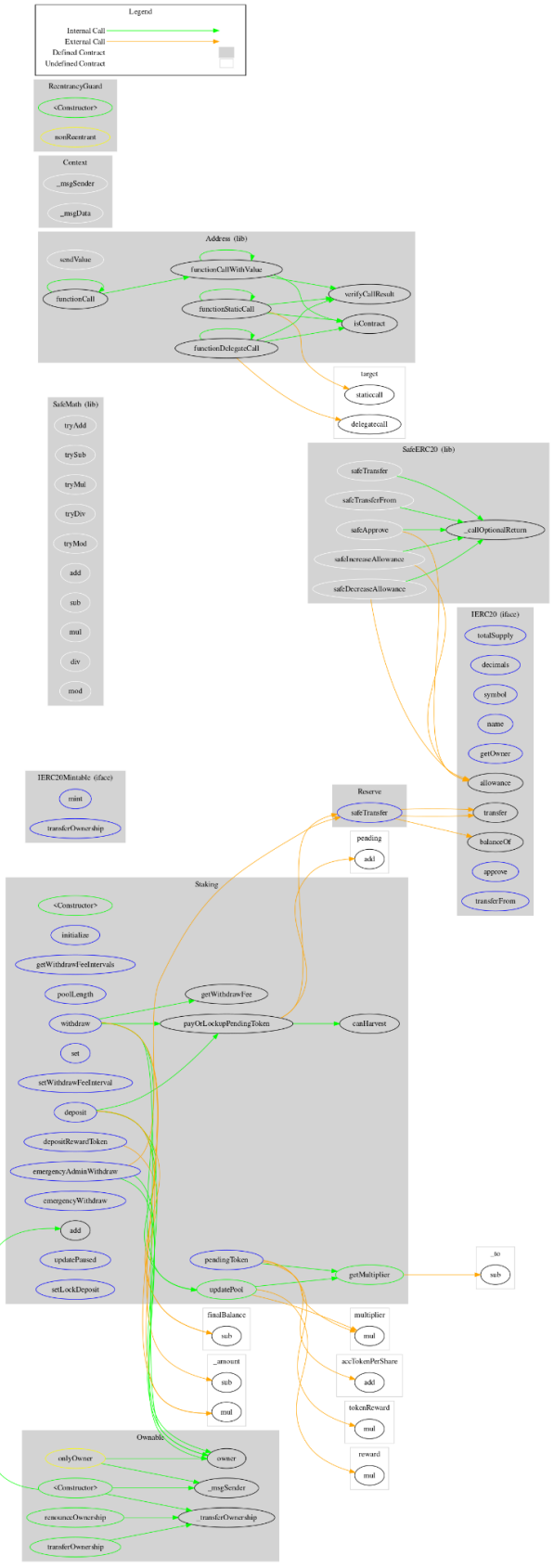
		Public	✓	-
Reserve	Implementation	Ownable		
	safeTransfer	External	✓	onlyOwner
Staking	Implementation	Ownable		
		Public	✓	-
	initialize	External	✓	onlyOwner
	getWithdrawFeeIntervals	External		-
	poolLength	External		-
	add	Public	✓	onlyOwner
	set	External	✓	onlyOwner
	setWithdrawFeeInterval	External	✓	onlyOwner
	getMultiplier	Public		-
	pendingToken	External		-
	canHarvest	Public		-
	updatePool	Public	✓	-
	depositRewardToken	External	✓	-
	deposit	External	✓	-
	withdraw	External	✓	-
	payOrLockupPendingToken	Internal	✓	
	getWithdrawFee	Public		-
	emergencyWithdraw	External	✓	-

	emergencyAdminWithdraw	External	✓	onlyOwner
	updatePaused	External	✓	onlyOwner
	setLockDeposit	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ReentrancyGuard
<Constructor>
nonReentrant
Context
_msgSender
_msgData
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
callOptionalReturn
safeIncreaseAllowance
safeDecreaseAllowance
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
allowance
transfer
balanceOf
approve
transferFrom
IERC20Mintable (iface)
mint
transferOwnership
Reserve
safeTransfer
pending
add
Staking
<Constructor>
initialize
getWithdrawFeeIntervals
poolLength
getWithdrawFee
withdraw
payOrLockupPendingToken
canHarvest
set
setWithdrawFeeInterval
deposit
depositRewardToken
emergencyAdminWithdraw
emergencyWithdraw
add
updatePassed
setLockDeposit
pendingToken
updatePool
getMultiplier
_to
sub
finalBalance
sub
multiplier
mul
_amount
sub
mul
accTokenPerShare
add
tokenReward
mul
reward
mul
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
transferOwnership
transferOwnership

Summary

Shiboost contract implements a staking and rewards mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

Shiboost

March 2023

Network	ETH
Address	0x4b922e3290ad813e86ab4624bc55fba3c361a82f
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Findings Breakdown 4
Analysis 5
ST - Stops Transactions 6
Description 6
Recommendation 6
OCTD - Transfers Contract's Tokens 7
Description 7
Recommendation 7
ELFM - Exceeds Fees Limit 8
Description 8
Recommendation 9
BC - Blacklists Addresses 10
Description 10
Recommendation 10
Diagnostics 11
PVC - Price Volatility Concern 12
Description 12
Recommendation 12
PTRP - Potential Transfer Revert Propagation 13
Description 13
Recommendation 13
RSML - Redundant SafeMath Library 14
Description 14
Recommendation 14
L02 - State Variables could be Declared Constant 15
Description 15
Recommendation 15
L04 - Conformance to Solidity Naming Conventions 16
Description 16
Recommendation 17
L07 - Missing Events Arithmetic 18
Description 18
Recommendation 18
L09 - Dead Code Elimination 19
Description 19

Recommendation 19
L16 - Validate Variable Setters 20
Description 20
Recommendation 20
L19 - Stable Compiler Version 21
Description 21
Recommendation 21
L20 - Succeeded Transfer Check 22
Description 22
Recommendation 22
Functions Analysis **23**
Inheritance Graph **30**
Flow Graph **31**
Summary **32**
Disclaimer **33**
About Cyberscope **34**

Review

Contract Name	Shiboost
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://etherscan.io/address/0x4b922e3290ad813e86ab4624bc55fba3c361a82f
Address	0x4b922e3290ad813e86ab4624bc55fba3c361a82f
Network	ETH
Symbol	Shiboost
Decimals	18
Total Supply	10,000,000

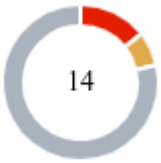
Audit Updates

Initial Audit	30 Mar 2023

Source Files

Filename	SHA256
Shiboost.sol	46823888f602c806bc29aa7e20bce8610e9b8f1327ccd4a79c50c6592f8b6a68

Findings Breakdown



Critical	2
Medium	1
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
	ST	Stops Transactions	Unresolved
	OCTD	Transfers Contract's Tokens	Unresolved
	OTUT	Transfers User's Tokens	Passed
	ELFM	Exceeds Fees Limit	Unresolved
	ULTW	Transfers Liquidity to Team Wallet	Passed
	MT	Mints Tokens	Passed
	BT	Burns Tokens	Passed
	BC	Blacklists Addresses	Unresolved

ST - Stops Transactions

Criticality	Critical
Location	Shiboost.sol#L607,626
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` and `_walletMax` to zero. As a result, the contract may operate as a honeypot.

```
if(!isTxLimitExempt[sender] && !isTxLimitExempt[recipient] &&
EnableTransactionLimit) {
    require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
}
...
if(checkWalletLimit && !isWalletLimitExempt[recipient]) {
    require(balanceOf(recipient).add(finalAmount) <= _walletMax,"Amount Exceed From
Max Wallet Limit!!");
}
```

Recommendation

The contract could embody a check for not allowing setting the _walletMax and _maxTxAmount less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	Shiboost.sol#L638
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueStuckedToken` function.

```
function rescueStuckedToken(address _token, uint _amount) external
onlyOwner {
    IERC20(_token).transfer(msg.sender,_amount);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	Shiboost.sol#L566,573
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setBuyTaxes` or `setSellTaxes` functions with a high percentage value.

```
function setBuyTaxes(uint _Liquidity, uint _Marketing , uint _Developer)
public onlyOwner {
    _buyLiquidityFee = _Liquidity;
    _buyMarketingFee = _Marketing;
    _buyDeveloperFee = _Developer;
    _totalTaxIfBuying =
_buyLiquidityFee.add(_buyMarketingFee).add(_buyDeveloperFee);
}

function setSellTaxes(uint _Liquidity, uint _Marketing , uint _Developer)
public onlyOwner {
    _sellLiquidityFee = _Liquidity;
    _sellMarketingFee = _Marketing;
    _sellDeveloperFee = _Developer;
    _totalTaxIfSelling =
_sellLiquidityFee.add(_sellMarketingFee).add(_sellDeveloperFee);
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Medium
Location	Shiboost.sol#L540
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `setBlacklist` function.

```
function setBlacklist(address _adr, bool _status) external onlyOwner {
    blacklist[_adr] = _status;
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PVC	Price Volatility Concern	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L16	Validate Variable Setters	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Shiboost.sol#L511
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setNumTokensBeforeSwap(uint256 newLimit) external onlyOwner() {
    minimumTokensBeforeSwap = newLimit;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Shiboost.sol#L676.680
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` and a `DeveloperWallet` as part of the transfer flow. These addresses can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketingWallet).transfer(amountETHMarketing);
payable(DeveloperWallet).transfer(amountETHDeveloper);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Shiboost.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, and overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change at https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Shiboost.sol#L323,324,325
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
string private _name = "Shiboost"
string private _symbol = "Shiboost"
uint8 private _decimals = 18
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Shiboost.sol#L150,151,167,186,328,334,343,344,345,347,348,349,351,352,358,359,369,495,527,540,566,573,638,744
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
address payable public DeveloperWallet =
payable(0x8754759f47Ea821E66801e9C654B5872d0F09e71)
mapping (address => uint256) _balances
uint256 public _buyLiquidityFee = 1
uint256 public _buyMarketingFee = 2
uint256 public _buyDeveloperFee = 2
uint256 public _sellLiquidityFee = 1
uint256 public _sellMarketingFee = 2
uint256 public _sellDeveloperFee = 2
uint256 public _totalTaxIfBuying
uint256 public _totalTaxIfSelling

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Shiboost.sol#L500,508,512,567,574
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_maxTxAmount = maxTxAmount
_walletMax  = newLimit
minimumTokensBeforeSwap = newLimit
_buyLiquidityFee = _Liquidity
_sellLiquidityFee = _Liquidity
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Shiboost.sol#L544
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function transferToAddressETH(address payable recipient, uint256 amount)
private {
        recipient.transfer(amount);
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Shiboost.sol#L516,520,524
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
marketingWallet = payable(newAddress)
liquidityReciever = payable(newAddress)
DeveloperWallet = payable(newAddress)
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Shiboost.sol#L7
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Shiboost.sol#L639
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(_token).transfer(msg.sender,_amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		

	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-

	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-

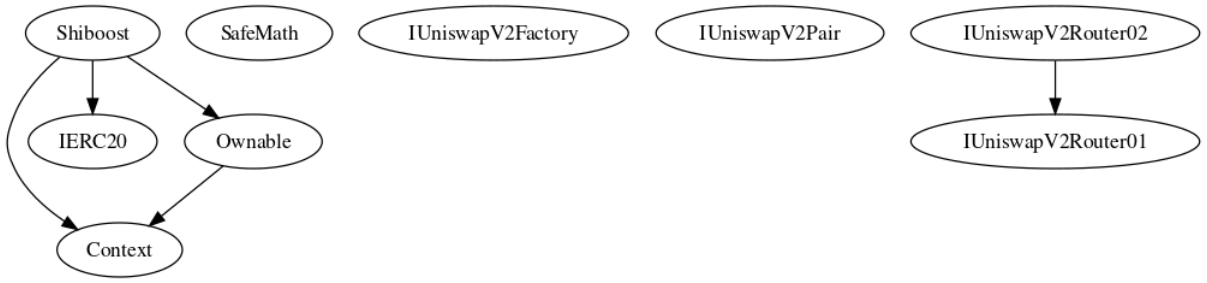
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-

IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Shiboost	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	approve	Public	✓	-
	_approve	Private	✓	
	setMarketPairStatus	Public	✓	onlyOwner
	setIsExcludedFromFee	Public	✓	onlyOwner

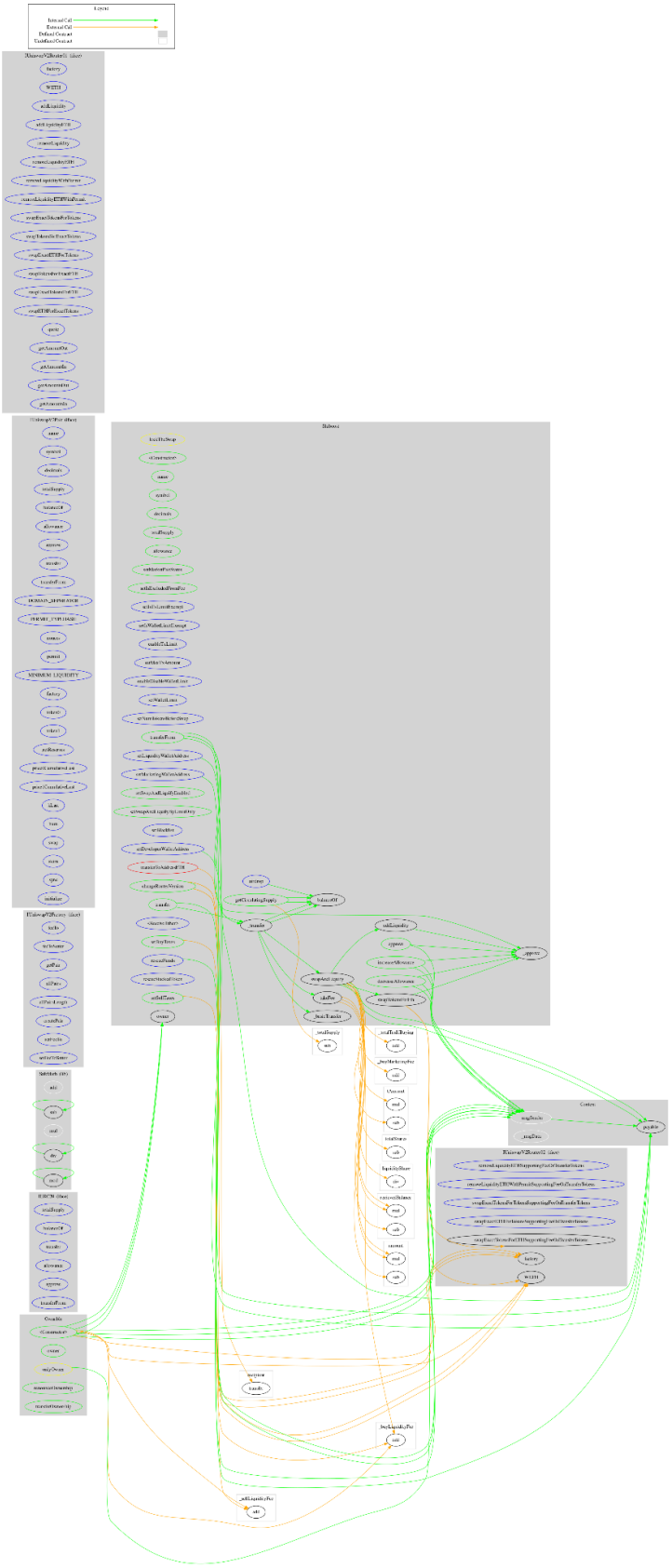
	setIsTxLimitExempt	External	✓	onlyOwner
	setIsWalletLimitExempt	External	✓	onlyOwner
	enableTxLimit	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	enableDisableWalletLimit	External	✓	onlyOwner
	setWalletLimit	External	✓	onlyOwner
	setNumTokensBeforeSwap	External	✓	onlyOwner
	setMarketingWalletAddress	External	✓	onlyOwner
	setLiquidityWalletAddress	External	✓	onlyOwner
	setDeveloperWalletAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setSwapAndLiquifyByLimitOnly	Public	✓	onlyOwner
	getCirculatingSupply	Public		-
	setBlacklist	External	✓	onlyOwner
	transferToAddressETH	Private	✓	
	changeRouterVersion	Public	✓	onlyOwner
	setBuyTaxes	Public	✓	onlyOwner
	setSellTaxes	Public	✓	onlyOwner
		External	Payable	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Private	✓	
	rescueStuckedToken	External	✓	onlyOwner

	rescueFunds	External	✓	onlyOwner
	_basicTransfer	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	takeFee	Internal	✓	
	airdrop	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Shiboost contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions, draining the contract's tokens, manipulating the fees, and blacklist addresses. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

Shiboost

March 2023

SHA256 0e4397499c71efd60c9ccf9e75964bf3c1ba1af39b0649a7c15dd16abed9b280

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Findings Breakdown **4**
Analysis **5**
ST - Stops Transactions 6
Description 6
Recommendation 6
OCTD - Transfers Contract's Tokens 7
Description 7
Recommendation 7
Diagnostics **8**
PTRP - Potential Transfer Revert Propagation 9
Description 9
Recommendation 9
PVC - Price Volatility Concern 10
Description 10
Recommendation 10
RSML - Redundant SafeMath Library 11
Description 11
Recommendation 11
L02 - State Variables could be Declared Constant 12
Description 12
Recommendation 12
L04 - Conformance to Solidity Naming Conventions 13
Description 13
Recommendation 14
L07 - Missing Events Arithmetic 15
Description 15
Recommendation 15
L09 - Dead Code Elimination 16
Description 16
Recommendation 16
L16 - Validate Variable Setters 17
Description 17
Recommendation 17
L19 - Stable Compiler Version 18
Description 18

Recommendation 18
L20 - Succeeded Transfer Check 19
Description 19
Recommendation 19
Functions Analysis **20**
Inheritance Graph **27**
Flow Graph **28**
Summary **29**
Disclaimer **30**
About Cyberscope **31**

Review

Contract Name	Shiboost
Testing Deploy	https://testnet.bscscan.com/address/0xf32840ba6ac967db25b711df3c727e471446e646
Symbol	SBT
Decimals	18
Total Supply	10,000,000

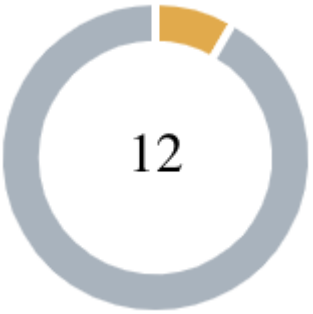
Audit Updates

Initial Audit	30 Mar 2023
Corrected Phase 2	02 Apr 2023

Source Files

Filename	SHA256
contracts/testingDeploy/shiboost.sol	0e4397499c71efd60c9ccf9e75964bf3c1ba1af39b0649a7c15dd16abed9b280

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Medium
Location	contracts/testingDeploy/shiboost.sol#L614,633
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` and `_walletMax` to zero.

```
if(!isTxLimitExempt[sender] && !isTxLimitExempt[recipient] &&
EnableTransactionLimit) {
    require(amount <= _maxTxAmount, "Transfer amount exceeds the
maxTxAmount.");
}
...
if(checkWalletLimit && !isWalletLimitExempt[recipient]) {
    require(balanceOf(recipient).add(finalAmount) <=
_walletMax,"Amount Exceed From Max Wallet Limit!!");
}
```

Recommendation

The contract could embody a check for not allowing setting the _maxTxAmount less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L645
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueStuckedToken` function.

```
function rescueStuckedToken(address _token, uint _amount) external
onlyOwner {
    IERC20(_token).transfer(msg.sender,_amount);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L16	Validate Variable Setters	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L682
Status	Unresolved

Description

The contract sends funds to `marketingWallet` and `DeveloperWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketingWallet).transfer(amountETHMarketing);
payable(DeveloperWallet).transfer(amountETHDeveloper);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achived by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L515
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `setNumTokensBeforeSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setNumTokensBeforeSwap(uint256 newLimit) external
onlyOwner() {
    minimumTokensBeforeSwap = newLimit;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases the gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L325,326,327,353
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
string private _name = "Shiboost"
string private _symbol = "SBT"
uint8 private _decimals = 18
uint256 public feeUnits = 100
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L152,153,169,188,330,336,345,346,347,349,350,351,355,356,362,363,373,499,531,570,578,645,751
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
address payable public DeveloperWallet =
payable(0x8754759f47Ea821E66801e9C654B5872d0F09e71)
mapping (address => uint256) _balances
uint256 public _buyLiquidityFee = 1
uint256 public _buyMarketingFee = 2
uint256 public _buyDeveloperFee = 3
uint256 public _sellLiquidityFee = 1
uint256 public _sellMarketingFee = 2
uint256 public _sellDeveloperFee = 3
uint256 public _totalTaxIfBuying
uint256 public _totalTaxIfSelling

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L504,512,516,571,579
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_maxTxAmount = maxTxAmount
_walletMax  = newLimit
minimumTokensBeforeSwap = newLimit
_buyLiquidityFee = _Liquidity
_sellLiquidityFee = _Liquidity
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L548
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function transferToAddressETH(address payable recipient, uint256
amount) private {
        recipient.transfer(amount);
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L520,524,528
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
marketingWallet = payable(newAddress)
liquidityReciever = payable(newAddress)
DeveloperWallet = payable(newAddress)
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L9
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/testingDeploy/shiboost.sol#L646
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(_token).transfer(msg.sender,_amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		

	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-

	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-

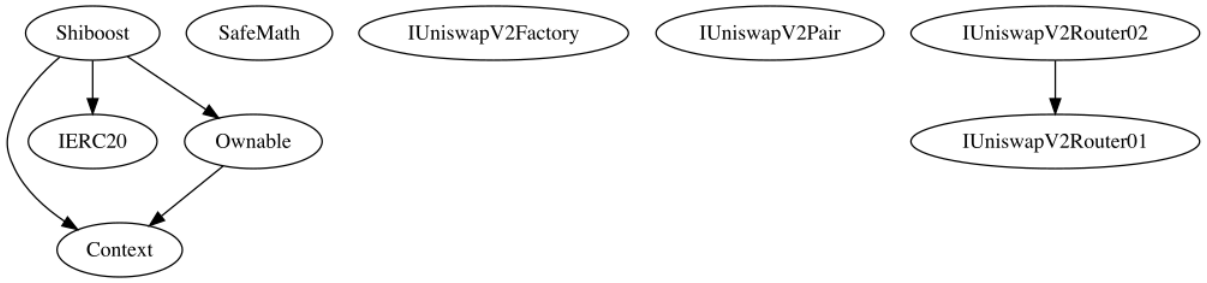
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-

IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Shiboost	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	approve	Public	✓	-
	_approve	Private	✓	
	setMarketPairStatus	Public	✓	onlyOwner
	setIsExcludedFromFee	Public	✓	onlyOwner

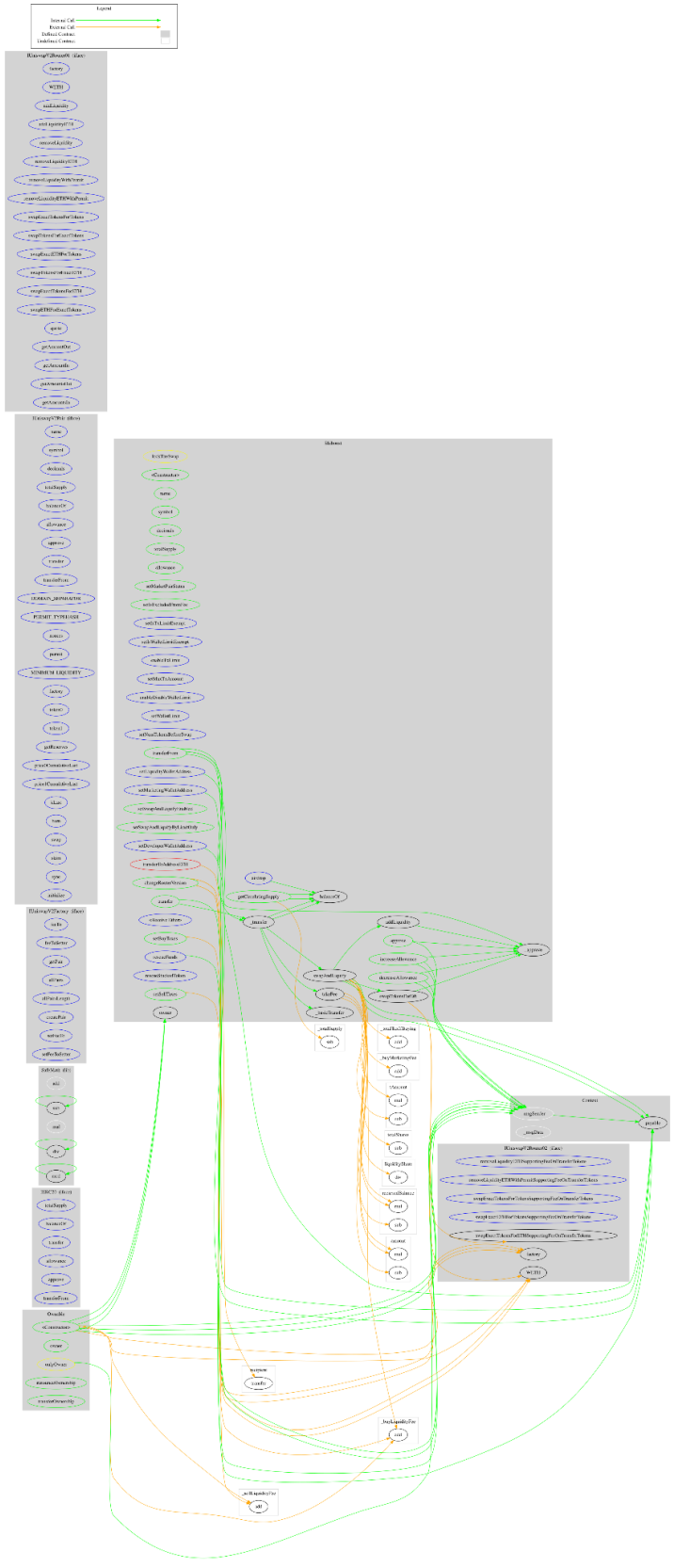
	setIsTxLimitExempt	External	✓	onlyOwner
	setIsWalletLimitExempt	External	✓	onlyOwner
	enableTxLimit	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	enableDisableWalletLimit	External	✓	onlyOwner
	setWalletLimit	External	✓	onlyOwner
	setNumTokensBeforeSwap	External	✓	onlyOwner
	setMarketingWalletAddress	External	✓	onlyOwner
	setLiquidityWalletAddress	External	✓	onlyOwner
	setDeveloperWalletAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setSwapAndLiquifyByLimitOnly	Public	✓	onlyOwner
	getCirculatingSupply	Public		-
	transferToAddressETH	Private	✓	
	changeRouterVersion	Public	✓	onlyOwner
	setBuyTaxes	Public	✓	onlyOwner
	setSellTaxes	Public	✓	onlyOwner
		External	Payable	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Private	✓	
	rescueStuckedToken	External	✓	onlyOwner
	rescueFunds	External	✓	onlyOwner

	_basicTransfer	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	takeFee	Internal	✓	
	airdrop	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Shiboost contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and drain the contract's tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 25% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

Shiboost

March 2023

SHA256 0b5879d79eff775c4db45384516e7a9e4e57f774a39e0d8588500712e432b203

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Introduction **4**
Roles 4
Owner 4
User 4
Findings Breakdown **5**
Diagnostics **6**
CO - Code Optimization 8
Description 8
Recommendation 8
PSU - Potential Subtraction Underflow 9
Description 9
Recommendation 9
NR - Non-Guaranteed Reward 10
Description 10
Recommendation 11
GO - Gas Optimization 12
Description 12
Recommendation 12
RV - Redundant Variable 13
Description 13
Recommendation 13
RI - Reward Inconsistency 14
Description 14
Recommendation 15
SS - State Synchronization 16
Description 16
Recommendation 16
RSML - Redundant SafeMath Library 17
Description 17
Recommendation 17
RSK - Redundant Storage Keyword 18
Description 18
Recommendation 18
IDI - Immutable Declaration Improvement 19
Description 19

Recommendation 19
L04 - Conformance to Solidity Naming Conventions 20
Description 20
Recommendation 21
L09 - Dead Code Elimination 22
Description 22
Recommendation 23
L17 - Usage of Solidity Assembly 24
Description 24
Recommendation 24
L19 - Stable Compiler Version 25
Description 25
Recommendation 25
L20 - Succeeded Transfer Check 26
Description 26
Recommendation 26
Functions Analysis **27**
Inheritance Graph **32**
Flow Graph **33**
Summary **34**
Disclaimer **35**
About Cyberscope **36**

Review

Testing Deploy	https://testnet.bscscan.com/address/0x76d3abd642f6a87ac9e7b01246e99c0d6e505cc3
Address	0x76d3AbD642F6a87aC9E7b01246e99c0d6E505cc3

Audit Updates

Initial Audit	10 Apr 2023

Source Files

Filename	SHA256
contracts/ShiboostStaking.sol	0b5879d79eff775c4db45384516e7a9e4e57f774a39e0d8588500712e432b203

Introduction

The ShiboostStaking implements a staking mechanism. Users can deposit tokens to a pool and receive rewards in the form of another token. The contract includes multiple pools, each with its own deposit and withdrawal fees, minimum deposit amount, and harvest interval. The contract also includes a mechanism for locking up rewards and penalizing early withdrawals. The contract owner can add and update pools, start and pause the staking platform, and emergency withdraw tokens from any pool.

Roles

Owner

The owner has authority over the following functions:

- `function initialize()`
- `function add(uint256 _tokenPerBlock, IERC20 _stakedToken, IERC20 _rewardToken, uint16 _depositFeeBP, uint256 _minDeposit, uint256 _harvestInterval, WithdrawFeeInterval[] memory withdrawFeeIntervals)`
- `function set(uint256 _pid, uint256 _tokenPerBlock, uint16 _depositFeeBP, uint256 _minDeposit, uint256 _harvestInterval)`
- `function setWithdrawFeeInterval(uint256 poolId, uint256 index, WithdrawFeeInterval memory _withdrawFeeInterval)`
- `function emergencyAdminWithdraw(uint256 _pid)`
- `function updatePaused(bool _value)`
- `function setLockDeposit(uint256 pid, bool locked)`

User

The user can interact with the following functions:

- `function getWithdrawFeeIntervals(uint256 poolId)`
- `function poolLength()`
- `function getMultiplier(uint256 _from, uint256 _to)`
- `function pendingToken(uint256 _pid, address _user)`
- `function canHarvest(uint256 _pid, address _user)`
- `function depositRewardToken(uint256 poolId, uint256 amount)`
- `function deposit(uint256 _pid, uint256 _amount)`
- `function withdraw(uint256 _pid, uint256 _amount)`
- `function getWithdrawFee(uint256 poolId, uint256 stakedTime)`
- `function emergencyWithdraw(uint256 _pid)`

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CO	Code Optimization	Unresolved
	PSU	Potential Subtraction Underflow	Unresolved
	NR	Non-Guaranteed Reward	Unresolved
	GO	Gas Optimization	Unresolved
	RV	Redundant Variable	Unresolved
	RI	Reward Inconsistency	Unresolved
	SS	State Synchronization	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSK	Redundant Storage Keyword	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L09	Dead Code Elimination	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved

●	L20	Succeeded Transfer Check	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L1002,1106,1115
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `balanceOf` function of an ERC20 token returns an `uint256` number, which means its value will always be greater than or equal to zero. The contract checks if its balance is greater than zero. This operation can be improved by using the `!=` operator.

```
pool.rewardToken.balanceOf(address(this)) > 0
pending > 0 || user.rewardLockedUp > 0
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. As described above, it is recommended to use the `!=` operator.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L1087
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

The `_withdrawFee` must be less than or equal to 1000 for the subtraction to work as expected. The owner has the authority to alter the fees of a certain poolId, and the `setWithdrawFeeInterval` function has no input validation. As a result, the operation may lead to an underflow.

```
uint256 _withdrawFee = getWithdrawFee(_pid, user.depositTimestamp);
uint256 feeAmount = _amount.mul(_withdrawFee).div(1000);
amountToTransfer = _amount.sub(feeAmount);
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

NR - Non-Guaranteed Reward

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L1141
Status	Unresolved

Description

The contract does not guarantee the users will get their rewards. The deposited amounts are transferred to the contract's balance, but the claimable rewards are transferred to the users from the `Reserve` contract. Additionally, the owner has the authority to claim all the reward funds from the `Reserve` contract.

```
function emergencyAdminWithdraw(uint256 _pid) external onlyOwner {
    PoolInfo storage pool = poolInfo[_pid];
    uint256 balanceToWithdraw = pool.rewardToken.balanceOf(address(this));
    require(balanceToWithdraw != 0, "STAKING: Not enough balance to
withdraw!");
    pool.rewardToken.transfer(owner(), balanceToWithdraw);
    rewardReserve.safeTransfer(pool.rewardToken, owner(),
pool.rewardToken.balanceOf(address(rewardReserve)));
    emit AdminEmergencyWithdraw(_pid,
pool.rewardToken.balanceOf(address(this)), pool.accTokenPerShare,
pool.tokenPerBlock, pool.lastRewardBlock);
    delete poolInfo[_pid];
}
```

Recommendation

The contract could guarantee the rewards will be available when a user deposits the amount. The owner is responsible for the `Reserve` contract having all the available funds so that users can claim their reward. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

GO - Gas Optimization

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L875
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `MAXIMUM_HARVEST_INTERVAL` and `harvestInterval` are declared as `uint256` variables. The `MAXIMUM_HARVEST_INTERVAL` is declared as a constant, hence its value cannot change. 14 days is equivalent to `14 * 24 * 60 * 60` seconds. This value can be stored in an `uint32` variable. Storing these variables as `uint256` takes up a lot more storage space, especially in the case of `harvestInterval`, which is a struct property stored in an array. As a result, the contract takes up more storage space than it actually requires, and has increased gas costs for transactions that include this variable.

```
uint256 public constant MAXIMUM_HARVEST_INTERVAL = 14 days;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RV - Redundant Variable

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L874
Status	Unresolved

Description

The `BONUS_MULTIPLIER` is a constant variable with a value of 1. It is only used for multiplication, hence the variable is redundant.

```
uint256 public constant BONUS_MULTIPLIER = 1;
```

Recommendation

The team is advised to remove this variable from the contract as it is redundant.

RI - Reward Inconsistency

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L828
Status	Unresolved

Description

The reward mechanism of the platform is not consistent. The `safeTransfer` function compares the reward amount with the Reserve contract's balance. If the amount is greater, then the user will receive a reward equal to the contract's balance. Since the owner has the authority to claim the balance of the Reserve contract, a user can claim the reward with the possibility of the reward amount the user receives is zero, if the owner previously claimed all the funds. The claiming process will be considered successful though. As a result, the reward mechanism is inconsistent.

```
function safeTransfer(
    IERC20 rewardToken,
    address _to,
    uint256 _amount
) external onlyOwner {
    uint256 tokenBal = rewardToken.balanceOf(address(this));
    if (_amount > tokenBal) {
        rewardToken.transfer(_to, tokenBal);
    } else {
        rewardToken.transfer(_to, _amount);
    }
}
```

Recommendation

The owner is responsible for the Reserve contract having all the available funds so that users can claim their reward. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

SS - State Synchronization

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L995
Status	Unresolved

Description

The contract's state is not being synchronized with what the user sees and the actual values. The `pendingToken` function will return the reward the user is actually entitled to, but in reality, the reward may be much less or even zero. This is a consequence of the RI and NR findings, which are described in detail below.

```
function pendingToken(uint256 _pid, address _user) external view returns
(uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];
    uint256 accTokenPerShare = pool.accTokenPerShare;
    if (
        block.number > pool.lastRewardBlock &&
        pool.stakedAmount != 0 &&
        pool.rewardToken.balanceOf(address(this)) > 0
    ) {
        uint256 multiplier = getMultiplier(pool.lastRewardBlock,
block.number);
        uint256 tokenReward = multiplier.mul(pool.tokenPerBlock);
        accTokenPerShare =
accTokenPerShare.add(tokenReward.mul(1e12).div(pool.stakedAmount));
    }
    return user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt);
}
```

Recommendation

The team is advised to consider all possible outcomes of the reward mechanism and provide realistic data to the users.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, and overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change at https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L996,997,1012,1123
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
PoolInfo storage pool
UserInfo storage user
WithdrawFeeInterval[] storage _withdrawFee
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L899
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
rewardReserve
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L830,831,928,929,930,931,932,933,967,968,969,970,971,984,991,995,1011,1016,1048,1076,1096,1131,1141,1151
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _to
uint256 _amount
uint256 _tokenPerBlock
IERC20 _stakedToken
IERC20 _rewardToken
uint16 _depositFeeBP
uint256 _minDeposit
uint256 _harvestInterval
uint256 _pid
WithdrawFeeInterval memory _withdrawFeeInterval
uint256 _to
uint256 _from
address _user
bool _value
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L393,418,447,480,490,507,517,598,614,623
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }

function functionCall(address target, bytes memory data) internal returns
(bytes memory) {
        return functionCall(target, data, "Address: low-level call
failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L371,546
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            size := extcodesize(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L2,102,112,342,560,660,687,764,826
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/ShiboostStaking.sol#L835,837,1145
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
rewardToken.transfer(_to, tokenBal)
rewardToken.transfer(_to, _amount)
pool.rewardToken.transfer(owner(), balanceToWithdraw)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Mintable	Interface	IERC20		
	mint	External	✓	-
	transferOwnership	External	✓	-
SafeMath	Library			
	tryAdd	Internal		

	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	

	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ReentrancyGuard	Implementation			

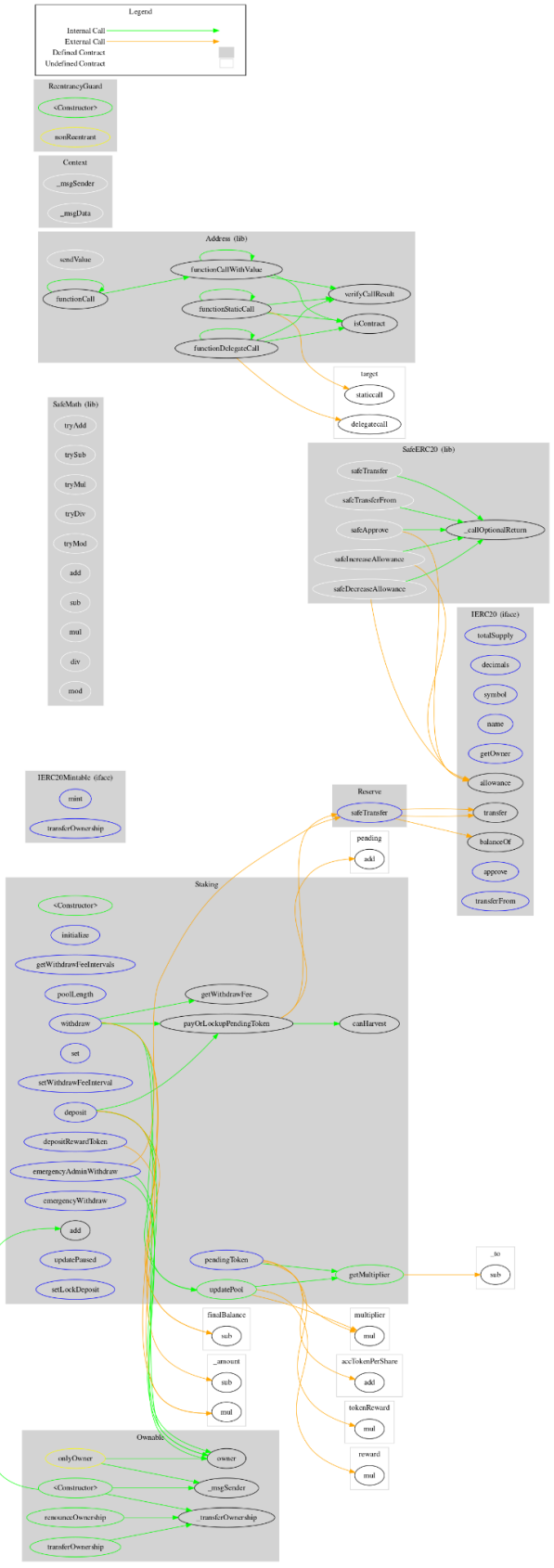
		Public	✓	-
Reserve	Implementation	Ownable		
	safeTransfer	External	✓	onlyOwner
Staking	Implementation	Ownable		
		Public	✓	-
	initialize	External	✓	onlyOwner
	getWithdrawFeeIntervals	External		-
	poolLength	External		-
	add	Public	✓	onlyOwner
	set	External	✓	onlyOwner
	setWithdrawFeeInterval	External	✓	onlyOwner
	getMultiplier	Public		-
	pendingToken	External		-
	canHarvest	Public		-
	updatePool	Public	✓	-
	depositRewardToken	External	✓	-
	deposit	External	✓	-
	withdraw	External	✓	-
	payOrLockupPendingToken	Internal	✓	
	getWithdrawFee	Public		-
	emergencyWithdraw	External	✓	-

	emergencyAdminWithdraw	External	✓	onlyOwner
	updatePaused	External	✓	onlyOwner
	setLockDeposit	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ReentrancyGuard
<Constructor>
nonReentrant
Context
_msgSender
_msgData
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
callOptionalReturn
safeIncreaseAllowance
safeDecreaseAllowance
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
allowance
transfer
balanceOf
approve
transferFrom
IERC20Mintable (iface)
mint
transferOwnership
Reserve
safeTransfer
pending
add
Staking
<Constructor>
initialize
getWithdrawFeeIntervals
poolLength
getWithdrawFee
withdraw
payOrLockupPendingToken
canHarvest
set
setWithdrawFeeInterval
deposit
depositRewardToken
emergencyAdminWithdraw
emergencyWithdraw
add
updatePassed
setLockDeposit
pendingToken
updatePool
getMultiplier
_to
sub
finalBalance
sub
multiplier
mul
_amount
sub
mul
accTokenPerShare
add
tokenReward
mul
reward
mul
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
transferOwnership
transferOwnership

Summary

Shiboost contract implements a staking and rewards mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io