Cyberscope

Audit Report

Scorpion

April 2024

Network BSC

Address 0x66cEdcA3990b9a96369A58d08d62963c4e4eC779

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DFD	Disproportional Fees Distribution	Unresolved
●	DMFC	Duplicate Marketing Fee Calculation	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	SCORP
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://bscscan.com/address/0x66cedca3990b9a96369a58d08d62963c4e4ec779
Address	0x66cedca3990b9a96369a58d08d62963c4e4ec779
Network	BSC
Symbol	SCORP
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

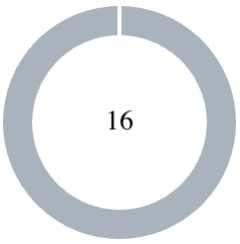
Audit Updates

Initial Audit	08 Apr 2024

Source Files

Filename	SHA256
SCORP.sol	258ed11f2fea4bdb116582ea8226fc068ac4209c67e7fafefa9f36e8bdeaba08

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	16

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	16	0	0	0

DFD - Disproportional Fees Distribution

Criticality	Minor / Informative
Location	SCORP.sol#L749
Status	Unresolved

Description

The smart contract applies fees, including marketing, burn, liquidity, and reflection fees, on token transfers. However, instead of sharing the initial fee amounts proportionally, it first divides the burn fee from the transferred amount and then applies the remaining fees to the reduced amount. This approach leads to a disproportional distribution of fees, potentially impacting the intended distribution model and investor expectations.

```
uint256 tokensForBurn =  amount.mul(burnFee).div(10**2);
_tokenTransfer(from, _burnAddress, tokensForBurn, false);
amount = amount.sub(tokensForBurn);
_tokenTransfer(from, to, amount, takeFee);
```

Recommendation

The team is adviced to review and revise the fee distribution mechanism to ensure proportional allocation according to the defined tokenomics. Consider implementing a consistent approach that distributes fees based on the initial transfer amount without prior deduction for specific fees.

DMFC - Duplicate Marketing Fee Calculation

Criticality	Minor / Informative
Location	SCORP.sol#L755
Status	Unresolved

Description

The smart contract contains a logic flaw resulting in the duplicate calculation and transfer of marketing fees. Specifically, when the accumulated contract amount reaches a certain threshold (numTokensSellToAddToLiquidity), the contract swaps tokens for ETH. However, instead of solely adding this ETH to the liquidity pool as intended, the contract also transfers a portion of this ETH to the marketing wallet. Consequently, the marketing wallet receives fees both in token form from every transfer and in ETH from the accumulated liquidity fees, resulting in a duplicate allocation of marketing fees.

```
function swapBack(uint256 contractBalance) private lockTheSwap {
    uint256 tokensForLiquidity = contractBalance.mul(liquidityFee).div(100);
    uint256 marketingTokens = contractBalance.mul(marketingFee).div(100);
    uint256 totalTokensToSwap = tokensForLiquidity + marketingTokens;
    ...
}
```

Such an inconsistency in fee distribution can dilute the intended distribution of fees, impacting the liquidity pool's growth. Moreover, investors may perceive this as a deviation from the tokenomics model.

Recommendation

To address this issue, it is recommended to thoroughly review the fee distribution logic, ensuring that marketing fees are only calculated and transferred from token transfers and not from accumulated liquidity fees. Implementing distinct mechanisms for allocating fees to the marketing wallet and liquidity pool, along with comprehensive testing and communication of the revised fee distribution mechanism, will help mitigate these concerns and foster a more transparent and sustainable ecosystem.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	SCORP.sol#L218,219,229
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_decimals
_tTotal
pancakeV2Pair
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	SCORP.sol#L395,437,438,442,443,484
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
_isExcluded[account] = true;
_isExcludedFromFee[account] = true;
_isExcludedFromBurnFee[account] = true;
_isExcludedFromFee[account] = false;
_isExcludedFromBurnFee[account] = false;
marketingWallet = _addr;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	SCORP.sol#L619,631
Status	Unresolved

Description

The contract is a missing transfer event emission when fees are transferred to the contract and marketing address as part of the transfer process. This omission can lead to a lack of visibility into fee transactions and hinder the ability of decentralized applications (DApps) like blockchain explorers to accurately track and analyze these transactions.

```
_rOwned[address(this)] = _rOwned[address(this)].add(
    rLiquidity
);
if (_isExcluded[address(this)])
    _tOwned[address(this)] = _tOwned[address(this)].add(
        tLiquidity
    );
...
_rOwned[marketingWallet] = _rOwned[marketingWallet].add(
    rMarketing
);
if (_isExcluded[marketingWallet])
    _tOwned[marketingWallet] = _tOwned[marketingWallet].add(
        tMarketing
    );
```

Recommendation

To address this issue, it is recommended to emit a transfer event after transferring the taxed amount to the correspinding addresses. The event should include relevant information such as the sender, recipient, and the amount transferred.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	SCORP.sol#L776
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForEth(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = pancakeV2Router .WETH();
    _approve(address(this), address(pancakeV2Router ), tokenAmount);
    pancakeV2Router .swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	SCORP.sol#L487
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 setNumTokensSellToAddToLiquidity(uint256 amount)
    external
    onlyOwner
{
    numTokensSellToAddToLiquidity = amount * 10 **_decimals;
}
```

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 exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	SCORP.sol#L201
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	SCORP.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, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	SCORP.sol#L437,438,442,443,495
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
_isExcludedFromFee[account] = true;
_isExcludedFromBurnFee[account] = true;
_isExcludedFromFee[account] = false;
_isExcludedFromBurnFee[account] = false;
swapAndLiquifyEnabled = _enabled;
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	SCORP.sol#L167
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.

```
address private _burnAddress = 0x000000000000000000000000000000000000dEaD
```

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	SCORP.sol#L128,483,494,636,640,648
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 WETH() external pure returns (address);
address _addr
bool _enabled
uint256 _amount
```

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	SCORP.sol#L455,473,491
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.

```
_sellRefFee = tFee
_buyRefFee = tFee
numTokensSellToAddToLiquidity = amount * 10 **_decimals
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	SCORP.sol#L753,754,760,767
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 tokensForLiquidity = contractBalance.mul(liquidityFee).div(100)
uint256 liquidityTokens = contractBalance * tokensForLiquidity /
totalTokensToSwap / 2
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	SCORP.sol#L484
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 = _addr
```

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	SCORP.sol#L3
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 ^0.8.20;
```

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.

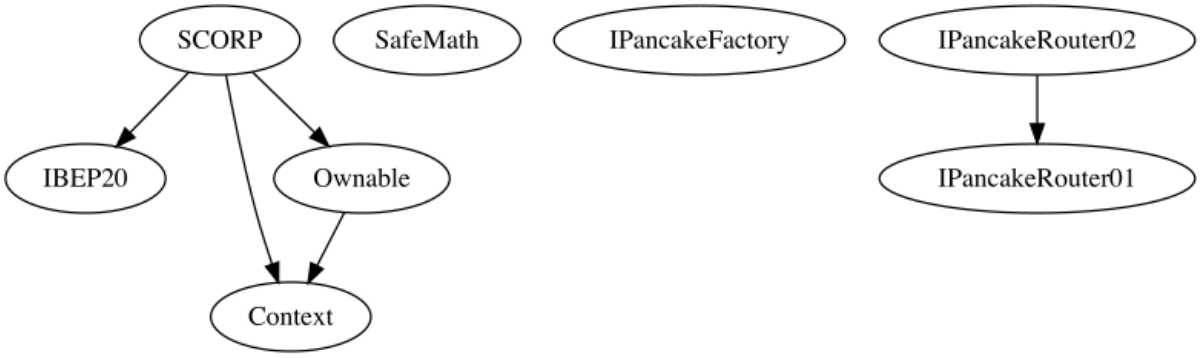
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SCORP	Implementation	Context, IBEP20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-

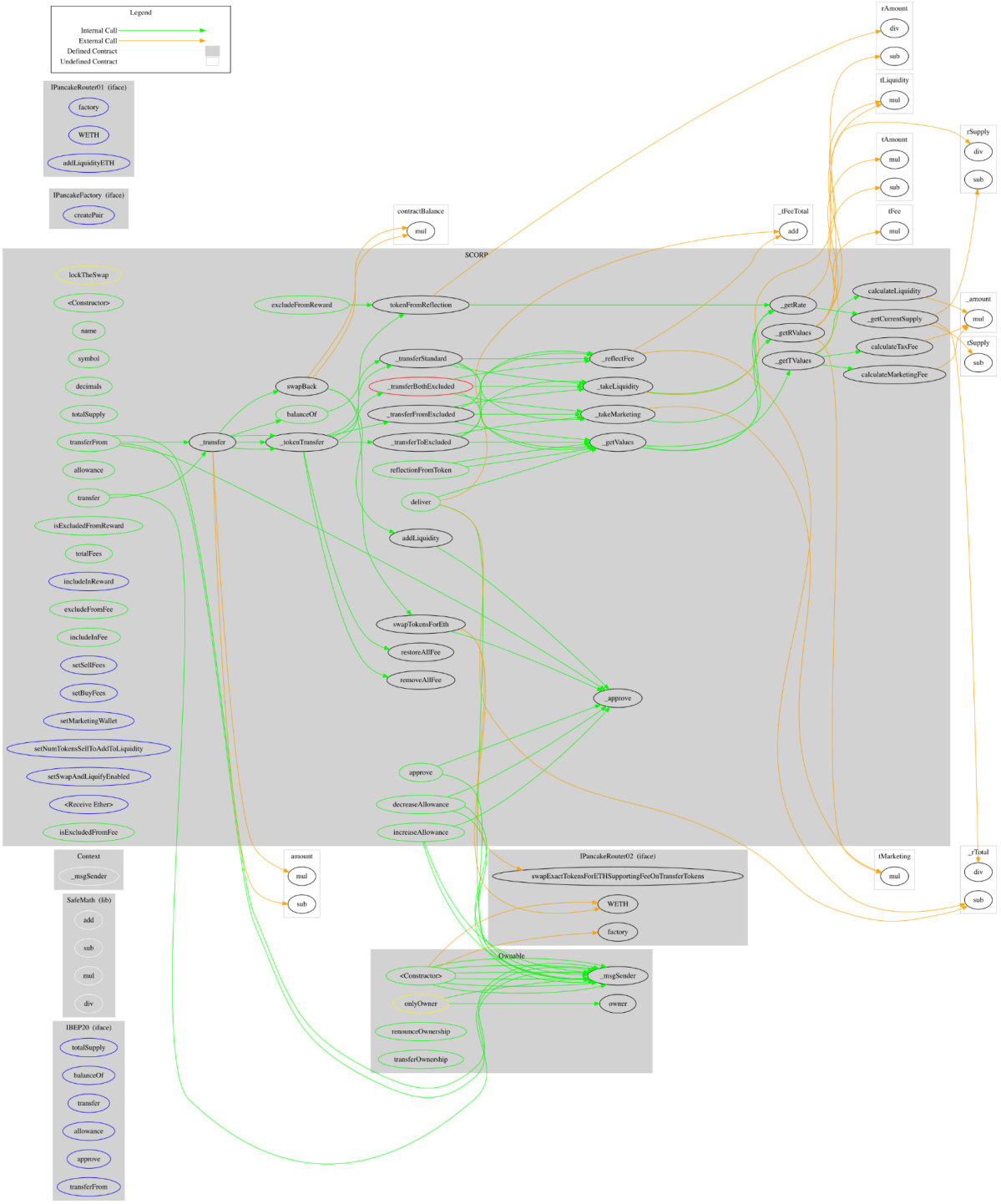
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	setBuyFees	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setNumTokensSellToAddToLiquidity	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	External	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	_takeMarketing	Private	✓	
	calculateTaxFee	Private		
	calculateMarketingFee	Private		
	calculateLiquidity	Private		

	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapBack	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IPancakeRouter01 (iface)
factory
WETH
addLiquidityETH
IPancakeFactory (iface)
createPair
SCORP
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
transferFrom
allowance
transfer
isExcludedFromReward
totalFees
includeInReward
excludeFromFee
includeInFee
setSellFees
setBuyFees
setMarketingWallet
setNumTokensSellToAddToLiquidity
setSwapAndLiquifyEnabled
<Receive Ether>
isExcludedFromFee
excludeFromReward
tokenFromReflection
swapBack
balanceOf
_transfer
_tokenTransfer
_transferStandard
_transferBothExcluded
_transferFromExcluded
_transferToExcluded
reflectionFromToken
deliver
addLiquidity
swapTokensForEth
restoreAllFee
removeAllFee
_approve
approve
decreaseAllowance
increaseAllowance
_reflectFee
_takeLiquidity
_takeMarketing
_getValues
_getRate
_getRValues
_getTValues
calculateLiquidity
_getCurrentSupply
calculateTaxFee
calculateMarketingFee
contractBalance
mul
_tFeeTotal
add
rAmount
div
sub
tLiquidity
mul
tAmount
mul
sub
tFee
mul
rSupply
div
sub
_amount
mul
tSupply
sub
Context
_msgSender
SafeMath (lib)
add
sub
mul
div
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
amount
mul
sub
IPancakeRouter02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
Ownable
<Constructor>
onlyOwner
renounceOwnership
transferOwnership
_msgSender
owner
tMarketing
mul
_rTotal
div
sub

Summary

Scorpion contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Scorpion is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 10% buy and sell 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