Cyberscope

Audit Report

Xtrink

May 2024

Network BSC

Address 0x13c2f83628D719FC94463e61D8270e3a38d80D9D

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	NPV	Non-Existent Pair Vulnerability	Unresolved
Minor / Informative	FRV	Fee Restoration Vulnerability	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	MEM	Misleading Error Messages	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	MTE	Missing Transfer Event	Unresolved
Minor / Informative	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
Minor / Informative	PMRM	Potential Mocked Router Manipulation	Unresolved
Minor / Informative	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor / Informative	RRS	Redundant Require Statement	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	RSD	Redundant Swap Duplication	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Token
Compiler Version	v0.8.6+commit.11564f7e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x13c2f83628D719FC94463e61D8270e3a38d80D9D
Address	0x13c2f83628D719FC94463e61D8270e3a38d80D9D
Network	BSC
Symbol	XTRINK
Decimals	10
Total Supply	10,000,000,000
Badge Eligibility	Must Fix Criticals

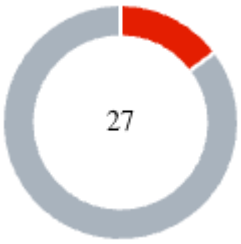
Audit Updates

Initial Audit	17 May 2024

Source Files

Filename	SHA256
Token.sol	0f3008b5dd7731a5197d4ddda7b7e1428cdc3ea05c24c87b492fc62f8fd1f99d

Findings Breakdown



Critical	4
Medium	0
Minor / Informative	23

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	0	0	0	0
Minor / Informative	23	0	0	0

ST - Stops Transactions

Criticality	Critical
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner, as described in detail in sections NPV, PMRM, and PTRP. As a result, the contract may operate as a honeypot.

Recommendation

The team is strongly encouraged to adhere to the recommendations outlined in the respective sections. By doing so, the contract can eliminate any potential of operating as a honeypot.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	Token.sol#L1189
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 `setAllFeePercent` function with a high percentage value.

```
function setAllFeePercent(uint8 taxFee, uint8 liquidityFee, uint8 burnFee,
uint8 walletFee, uint8 buybackFee, uint8 walletCharityFee, uint8
rewardFee) external onlyOwner() {
    require(taxFee >= 0 && taxFee <=maxTaxFee,"TF err");
    require(liquidityFee >= 0 && liquidityFee <=maxLiqFee,"LF err");
    require(burnFee >= 0 && burnFee <=maxBurnFee,"BF err");
    require(walletFee >= 0 && walletFee <=maxWalletFee,"WF err");
    require(buybackFee >= 0 && buybackFee <=maxBuybackFee,"BBF err");
    require(walletCharityFee >= 0 && walletCharityFee <=maxWalletFee,"WFT
err");
    require(rewardFee >= 0 && rewardFee <=maxTaxFee,"RF err");
    //both tax fee and reward fee cannot be set
    require(rewardFee == 0 || taxFee == 0,"RT fee err");
    _taxFee = taxFee;
    _liquidityFee = liquidityFee;
    _burnFee = burnFee;
    _buybackFee = buybackFee;
    _walletFee = walletFee;
    _walletCharityFee = walletCharityFee;
    _rewardFee = rewardFee;
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Critical
Location	Token.sol#L2007
Status	Unresolved

Description

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

```
function blacklistAddress(address account, bool value) external onlyOwner
{
    _isBlacklisted[account] = value;
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

NPV - Non-Existent Pair Vulnerability

Criticality	Critical
Location	Token.sol#L1576
Status	Unresolved

Description

As part of the transfer flow, the contract attempts to swap a percentage of its tokens for reward tokens when the reward fee is greater than zero. However, the `rewardToken` was initialized to the zero address and there is no administrative function to modify its value. As a result, when the `swapTokensForRewardToken` function is invoked, the transaction will revert due to the non-existent trading pair between WETH and the zero address.

```
function swapTokensForRewardToken(uint256 tokenAmount) private {
    address[] memory path = new address[](3);
    path[0] = address(this);
    path[1] = pcsV2Router.WETH();
    path[2] = rewardToken;

    _approve(address(this), address(pcsV2Router), tokenAmount);

    // make the swap
    pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp.add(300)
    );
}
```

Recommendation

The team is strongly advised to either introduce an administrative function to properly set the `rewardToken` address after contract deployment or permanently set the reward fee to zero. These actions will ensure that the contract operates smoothly without interruptions in the transfer flow.

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	Token.sol#L1612,1627
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
if(!takeFee)
    removeAllFee();

if(!takeFee)
    restoreAllFee();
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Token.sol#L998,999,1009,1019
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
rewardToken
numTokensSellToAddToLiquidity
```

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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	Token.sol#L1701,1767,1803
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_tDividendTotal > 0)
require(false)
require(!excludedFromDividends[account])
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Token.sol#L1074,1182,1186,1237,1242,1246,1250
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.

```
pcsV2Router = _pcsV2Router;
_isExcludedFromFee[account] = true;
_isExcludedFromFee[account] = false;
feeWallet = newFeeWallet;
feeWalletCharity = newFeeWallet;
walletFeeInBNB = inBNB;
walletCharityFeeInBNB = inBNB;
```

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.

MTE - Missing Transfer Event

Criticality	Minor / Informative
Location	Token.sol#L1308
Status	Unresolved

Description

According to the ERC20 token standard, when tokens are created, issued, or transferred, a Transfer event should be emitted. This ensures transparency and allows for easy tracking of token transfer events by external systems or interfaces. However, the contract does not emit a Transfer event in several token transfers. This deviation from the ERC20 standard can lead to potential issues in token tracking and integration with other platforms or services that rely on the standard Transfer event for token transfers.

```
_rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
```

Recommendation

It is recommended to adhere to the ERC20 standard by emitting Transfer events whenever tokens are transferred or created. Any function that creates, burns or transfers tokens should emit the appropriate Transfer event to ensure full compliance with the ERC20 standard and enhance transparency.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Token.sol#L1543
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 swapTokensForBNB(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = pcsV2Router.WETH();

    _approve(address(this), address(pcsV2Router), tokenAmount);

    // make the swap
    pcsV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        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.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	Token.sol#L1063
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
function updatePcsV2Router(address newAddress) public onlyOwner {
    require(
    newAddress != address(pcsV2Router),
    "The router already has that address"
    );
    IUniswapV2Router02 _pcsV2Router = IUniswapV2Router02(newAddress);
        // Create a uniswap pair for this new token
    pcsV2Pair = IUniswapV2Factory(_pcsV2Router.factory())
        .createPair(address(this), _pcsV2Router.WETH());

    // set the rest of the contract variables
    pcsV2Router = _pcsV2Router;
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Token.sol#L1469,1494
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` 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.

```
transferEth(feeWallet, newBalance);
transferEth(feeWalletCharity, newBalance);
```

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.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	Token.sol#L101
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Token.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	Token.sol#L1182,1186,1231,1246,1250
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;
_isExcludedFromFee[account] = false;
swapAndLiquifyEnabled = _enabled;
walletFeeInBNB = inBNB;
walletCharityFeeInBNB = inBNB;
```

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.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	Token.sol#L1466,1476,1491,1522
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
swapTokensForBNB(spentAmount);
swapTokensForBNB(half);
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Token.sol#L846,848,849,850,851,852,853,854,899,908
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 dead = 0x000000000000000000000000000000000000dEaD
uint8 public maxLiqFee = 10
uint8 public maxTaxFee = 10
uint8 public maxBurnFee = 10
uint8 public maxWalletFee = 10
uint8 public maxBuybackFee = 10
uint8 public minMxTxPercentage = 1
uint8 public minMxWalletPercentage = 1
address public router = 0x10ED43C718714eb63d5aA57B78B54704E256024E
bool public mintedByUnicarve = true
```

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	Token.sol#L699,861,862,904,910,911,914,917,920,923,926,929,932,946,947,951,1230,1253,1313,1319,1740,1744,1748,1752,1835
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);
uint256 public _tDividendTotal = 0
uint256 internal constant magnitude = 2**128
uint256 public _tTotal
string public _name
string public _symbol
uint8 public _taxFee = 0
uint8 public _rewardFee = 0
uint8 public _liquidityFee = 0
uint8 public _burnFee = 0
uint8 public _walletFee = 0
uint8 public _walletCharityFee = 0
uint8 public _buybackFee = 0
uint256 public _maxTxAmount

...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Token.sol#L250
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Token.sol#L1199,1213,1218,1225,1255,1999
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.

```
_taxFee = taxFee
buyBackUpperLimit = buyBackLimit * 10**18

_maxTxAmount = _tTotal.mul(maxTxPercent).div(
            10**4
        )

_maxWalletAmount = _tTotal.mul(maxWalletPercent).div(
            10**4
        )
minimumTokenBalanceForDividends = _minimumTokenBalanceForDividends
gasForProcessing = newValue
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	Token.sol#L1038,1039,1040,1041,1042,1043,1044,1190,1191,1192,1193,1194,1195,1196
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(fee.setTaxFee >= 0 && fee.setTaxFee <=maxTaxFee,"TF err")
require(fee.setLiqFee >= 0 && fee.setLiqFee <=maxLiqFee,"LF err")
require(fee.setBurnFee >= 0 && fee.setBurnFee <=maxBurnFee,"BF err")
require(fee.setWalletFee >= 0 && fee.setWalletFee <=maxWalletFee,"WF err")
require(fee.setBuybackFee >= 0 && fee.setBuybackFee <=maxBuybackFee,"BBF
err")
require(fee.setWalletCharityFee >= 0 && fee.setWalletCharityFee
<=maxWalletFee,"WFT err")
require(fee.setRewardFee >= 0 && fee.setRewardFee <=maxTaxFee,"RF err")
require(taxFee >= 0 && taxFee <=maxTaxFee,"TF err")
require(liquidityFee >= 0 && liquidityFee <=maxLiqFee,"LF err")
require(burnFee >= 0 && burnFee <=maxBurnFee,"BF err")
require(walletFee >= 0 && walletFee <=maxWalletFee,"WF err")
require(buybackFee >= 0 && buybackFee <=maxBuybackFee,"BBF err")
require(walletCharityFee >= 0 && walletCharityFee <=maxWalletFee,"WFT
err")
require(rewardFee >= 0 && rewardFee <=maxTaxFee,"RF err")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Token.sol#L296,331,405,432,458,468,483,493,498,534,538,549,560,565,576,1762
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function get(Map storage map, address key) internal view returns (uint) {
        return map.values[key];
    }

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Token.sol#L1451,1457,1475,1481,1500
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.

```
spentAmount = contractTokenBalance.div(totFee).mul(_walletFee)
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Token.sol#L1740,1744,1748,1752
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Token.sol#L1009
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.

```
rewardToken = _rewardToken
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Token.sol#L412,511
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 { codehash := extcodehash(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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Token.sol#L1696
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(tokenAddress).transfer(owner(), tokenAmount)
```

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		-
	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		

Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IterableMapping	Library			
	get	Internal		
	getIndexOfKey	Internal		
	getKeyAtIndex	Internal		
	size	Internal		
	set	Internal	✓	
	remove	Internal	✓	

Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	geUnlockTime	Public		-

	lock	Public	✓	onlyOwner
	unlock	Public	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	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	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Token	Implementation	Context, IERC20, Ownable		
		Public	Payable	-
	name	Public		-

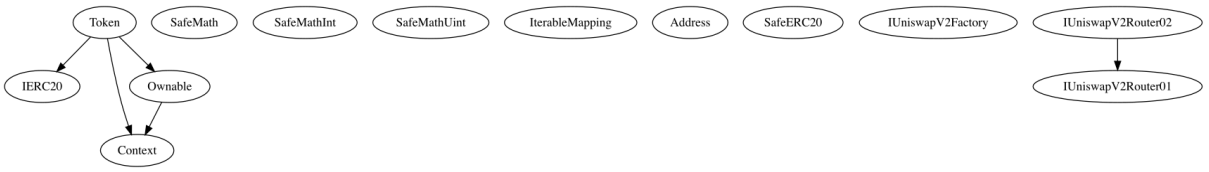
	updatePcsV2Router	Public	✓	onlyOwner
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setAllFeePercent	External	✓	onlyOwner
	buyBackUpperLimitAmount	Public		-
	setBuybackUpperLimit	External	✓	onlyOwner

	setMaxTxPercent	External	✓	onlyOwner
	setMaxWalletPercent	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setFeeWallet	External	✓	onlyOwner
	setFeeWalletCharity	External	✓	onlyOwner
	setWalletFeeTokenType	External	✓	onlyOwner
	setWalletCharityFeeTokenType	External	✓	onlyOwner
	setMinimumTokenBalanceForDividends	External	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	calculateTaxFee	Private		
	calculateLiquidityFee	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	

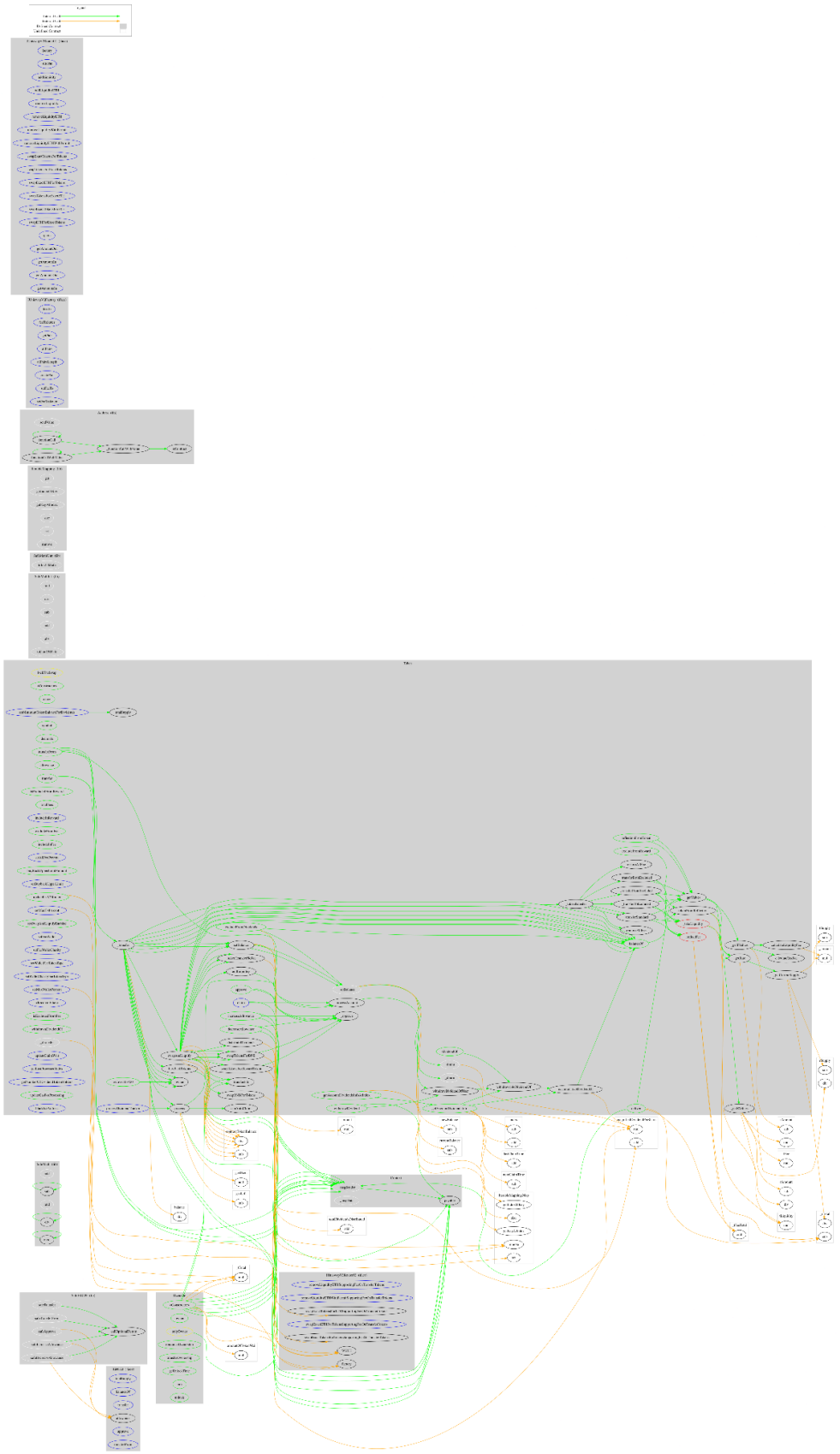
	swapAndLiquify	Private	✓	lockTheSwap
	buyBackTokens	Private	✓	lockTheSwap
	swapTokensForBNB	Private	✓	
	swapBNBForTokens	Private	✓	
	swapTokensForRewardToken	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	
	_transferBothExcluded	Private	✓	
	_tokenTransferNoFee	Private	✓	
	transferEth	Private	✓	
	recoverBEP20	Public	✓	onlyOwner
	distributeDividends	Internal	✓	
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_dtransfer	Internal	✓	
	_dmint	Internal	✓	

	_dburn	Internal	✓	
	_setBalance	Internal	✓	
	excludeFromDividends	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	getAccountDividendsInfo	Public		-
	getAccountDividendsInfoAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	Private	✓	
	process	Public	✓	-
	processAccount	Internal	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	processDividendTracker	External	✓	-
	blacklistAddress	External	✓	onlyOwner
	claim	External	✓	-

Inheritance Graph



Flow Graph



Summary

Xtrink 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, manipulate the fees and massively blacklist addresses. 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