Cyberscope

Audit Report

Triton

January 2024

Network BSC

Address 0x7b3fa14506395469041948af8cb24309b32ffcd2

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	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	SFO	Sell Fee Oversight	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RES	Redundant Event Statement	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	RTCI	Reward Token Change Inconsistency	Unresolved
●	UCL	Unoptimized Calculation Logic	Unresolved

●	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
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	TRITON
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x7b3fa14506395469041948af8cb24309b32ffcd2
Address	0x7b3fa14506395469041948af8cb24309b32ffcd2
Network	BSC
Symbol	TTN
Decimals	18
Total Supply	10,000,000,000
Badge Eligibility	Must Fix Criticals

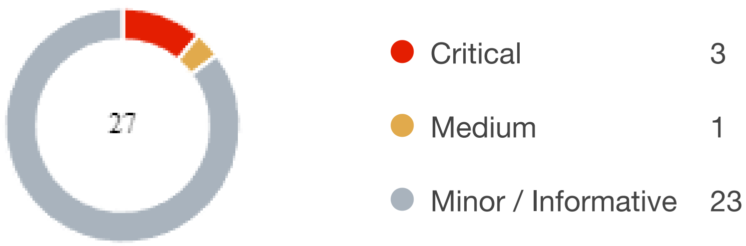
Audit Updates

Initial Audit	09 Jan 2024

Source Files

Filename	SHA256
TRITON.sol	f9abead4009cb03ac5fb70db4d1aafe88ef6a9e2098a7d3f92339effb77ab8e1

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	1	0	0	0
Minor / Informative	23	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	TRITON.sol#L1453
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if (!canTransferBeforeTradingIsEnabled[from]) {
    require(tradingEnabled, "Trading has not yet been
enabled");
}
```

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 minimum 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.

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.

ELFM - Exceeds Fees Limit

Criticality	Critical
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 updateFees function with a high percentage value. Specifically, the `totalSellFees` and `totalBuyFees` variables doesn't account for the `buyDeadFees` and `sellDeadFees` fee variables.

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

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

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

    require(totalSellFees <= 21 && totalBuyFees <= 21,
"total");
...
```

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.

TSD - Total Supply Diversion

Criticality	Critical
Location	TRITON.sol#L1561
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

```
if (deadFees > 0) {
    burntokens = amount.mul(deadFees) / 100;
    super._transfer(from, DEAD, burntokens);
    _totalSupply = _totalSupply.sub(burntokens);

    }
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

SFO - Sell Fee Oversight

Criticality	Medium
Location	TRITON.sol#L1475
Status	Unresolved

Description

The contract contains the `stakingEnabled` variable that, when enabled, does not account for sell fees within the `else if (isSelling) condition`. This oversight in the contract logic could lead to inconsistencies in fee application and enforcement, particularly in scenarios involving selling transactions. The current implementation under the `stakingEnabled` condition focuses on checking and resetting the staking status of tokens, but it neglects the sets of sell fees, which are critical for maintaining the economic balance and incentivization structure of the contract.

```
else if (!isBuying && stakingEnabled) {
    require(
        stakingUntilDate[from] <= block.timestamp,
        "Tokens are staked and locked!"
    );
    if (stakingUntilDate[from] != 0) {
        stakingUntilDate[from] = 0;
        stakingBonus[from] = 0;
    }
}

else if (isSelling) {
    RewardsFee = sellRewardsFee;
    deadFees = sellDeadFees;
    marketingFees = sellMarketingFees;
    liquidityFee = sellLiquidityFee;
    devFees = sellDevFee;

    if (limitsInEffect) {
    require(block.timestamp >=
_holderLastTransferTimestamp[tx.origin] + cooldowntimer,
            "cooldown period active");
    _holderLastTransferTimestamp[tx.origin] = block.timestamp;

    }
```

Recommendation

It is recommended to revise the contract logic to ensure that sell fees are appropriately applied even when the `stakingEnabled` variable is active. This can be achieved by setting the sell fee logic, regardless of the state of the `stakingEnabled` variable. By doing so, the contract will consistently apply sell fees for all selling transactions, maintaining the intended economic mechanisms and fee structures. This adjustment will enhance the fairness and predictability of the contract's behavior, ensuring that all participants are subject to the same fee rules irrespective of their staking status. Additionally, this change will help prevent potential exploits or imbalances that could arise from the current oversight.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	TRITON.sol#L1466,1519,1729
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code. The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
uint256 tFees = amount.mul(transferFee).div(100);
uint256 totalFees = RewardsFee.add(liquidityFee + marketingFees
+ devFees);
uint256 devPayout = buyDevFee.add(sellDevFee) * feePortions;
...
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	TRITON.sol#L1675
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 fromBuy = tokens.mul(buyAmount).div(totalAmount);
uint256 fromSell = tokens.mul(sellAmount).div(totalAmount);
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	TRITON.sol#L793,881,1126,1134,1198,1225,1256,1909,1915,1957
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(totalSupply() > 0)
require(false)
require(stakingAmounts[duration] != bonus)
require(!tradingEnabled)
require(stakingEnabled != enable)
require(swapAndLiquifyEnabled != enabled)
require(newValue >= 200000 && newValue <= 1000000)
require(allowCustomTokens != allow)
require(allowAutoReinvest != allow)
require(dividendsPaused != value)
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	TRITON.sol#L1224,1524
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 setSwapTriggerAmount(uint256 amount) public onlyOwner
{
        swapTokensAtAmount = amount * (10**18);
    }
    ...
    bool canSwap = contractTokenBalance >= swapTokensAtAmount;

    if (canSwap && !automatedMarketMakerPairs[from]) {
        swapping = true;

        if (swapAndLiquifyEnabled && liquidityFee > 0 &&
totalBuyFees > 0) {
            ...

            uint256 swapTokens =
swapSellTokens.add(swapBuyTokens);

            swapAndLiquify(swapTokens);
        }
...
```

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.

RES - Redundant Event Statement

Criticality	Minor / Informative
Location	TRITON.sol#L997,1002
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 contract contains some events statements that are not used in the contract's implemantation.

```
event UpdateDividendTracker(
    address indexed newAddress,
    address indexed oldAddress
);

event UpdateUniswapV2Router(
    address indexed newAddress,
    address indexed oldAddress
);
```

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. It is recommend removing the unused event statement from the contract.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	TRITON.sol#L413
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	TRITON.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.

```
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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	TRITON.sol#L1924,1934,1938
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.

```
function excludeFromDividends(address account) external
onlyOwner {
    //require(!excludedFromDividends[account]);
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}

function includeFromDividends(address account) external
onlyOwner {
    excludedFromDividends[account] = false;
}

function setAutoClaim(address account, bool value) external
onlyOwner {
    excludedFromAutoClaim[account] = value;
}

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

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	TRITON.sol#L1548,1610
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.

```
	swapAndLiquify(swapTokens);
	 }
	 uint256 remainingBalance = balanceOf(address(this));
	 swapAndSendDividends(remainingBalance);
	 ...
	 function swapAndLiquify(uint256 tokens) private {
	 ...
	 swapTokensForEth(half);
	 ...
	 }
	 function swapAndSendDividends(uint256 tokens) private {
	 ...
	 swapTokensForEth(tokens);
	 ...
	 }
```

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.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	TRITON.sol#L1674,1689
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
uint256 totalAmount = buyAmount.add(sellAmount);
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

RTCI - Reward Token Change Inconsistency

Criticality	Minor / Informative
Location	TRITON.sol#L2157
Status	Unresolved

Description

The contract is currently designed to allow the contract owner to change the distribution token address using the `updatePayoutToken` method. While this feature offers flexibility, a key concern is the requirement for a valid pair address between the native token and the new token address to ensure seamless transactions. Additionally, changing the reward token address without resetting the internal state of the distributor can lead to discrepancies, as the variables and calculations are based on the balance and characteristics of the previous token. This could result in inaccurate reward distributions or even failures in the distribution process.

```
function updatePayoutToken(address token) public onlyOwner {
    defaultToken = token;
}
```

Recommendation

It is recommended to carefully evaluate the necessity and implications of allowing the reward token address to be changed. If retaining this feature, the team should implement safeguards and additional logic to address the potential side-effects of such changes. This includes verifying the existence of a valid pair address between the native token and the new token. Alternatively, considering the removal of the option to change the reward token address could provide a more stable and predictable reward mechanism, thereby enhancing the reliability and trustworthiness of the contract.

UCL - Unoptimized Calculation Logic

Criticality	Minor / Informative
Location	TRITON.sol#L1529
Status	Unresolved

Description

The contract is currently structured to perform separate calculations for buy and sell fees, subsequently adding these amounts together to determine the total swap tokens. Specifically, it calculates `swapAmountBought` and `swapAmountSold` based on the `buyAmount` and `sellAmount` respectively, and then applies the liquidity fee to each to obtain `swapBuyTokens` and `swapSellTokens` . These two values are then added to get the total `swapTokens` . This approach, while functional, introduces unnecessary complexity and redundancy in calculations, especially considering that the same end result could be achieved with a more streamlined process.

```
if (swapAndLiquifyEnabled && liquidityFee > 0 && totalBuyFees >
0) {
    uint256 totalBuySell = buyAmount.add(sellAmount);
    uint256 swapAmountBought = contractTokenBalance
        .mul(buyAmount)
        .div(totalBuySell);
    uint256 swapAmountSold = contractTokenBalance
        .mul(sellAmount)
        .div(totalBuySell);

    uint256 swapBuyTokens = swapAmountBought
        .mul(liquidityFee)
        .div(totalBuyFees);

    uint256 swapSellTokens = swapAmountSold
        .mul(liquidityFee)
        .div(totalSellFees);

    uint256 swapTokens = swapSellTokens.add(swapBuyTokens);

    swapAndLiquify(swapTokens);
}
```

Recommendation

It is recommended to simplify the calculation process by applying the liquidity fee calculation directly to the `totalBuySell` value, which represents the sum of `buyAmount` and `sellAmount`. This approach would eliminate the need for separate calculations for buy and sell amounts, thereby reducing computational overhead and potential for errors. By directly using the `totalBuySell` value, the contract can more efficiently calculate the total `swapTokens` needed for the `swapAndLiquify` function. This optimization not only streamlines the code but also potentially reduces gas costs associated with these calculations, enhancing the overall efficiency and performance of the contract.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	TRITON.sol#L928
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 public DEAD =
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	TRITON.sol#L53,55,86,471,563,775,840,844,853,862,928,1178,1280,1281,1742,1743,1821,1970
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 (uint256);
address public _owner
function WETH() external pure returns (address);
uint256 internal constant magnitude = 2**128
address _owner
address public DEAD =
0x000000000000000000000000000000000000dEaD
uint256 GWEI
uint256 RewardsBuy
uint256 RewardsSell
address[] memory _contributors
uint256[] memory _balances
TRITON public TRITONContract

...
```

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	TRITON.sol#L510
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	TRITON.sol#L1180,1186,1193,1221,1676
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.

```
gasPriceLimit = GWEI * 1 gwei
cooldowntimer = value
maxWallet = value
swapTokensAtAmount = amount * (10**18)
buyAmount = buyAmount.sub(fromBuy)
```

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	TRITON.sol#L541,811,876,2280
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;
    }

...
```

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	TRITON.sol#L1527,1530,1534,1538,1696,1704
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 swapAmountBought = contractTokenBalance
                            .mul(buyAmount)
                            .div(totalBuySell)
uint256 swapBuyTokens = swapAmountBought
                            .mul(liquidityFee)
                            .div(totalBuyFees)
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	TRITON.sol#L1443,1444,1445,1446,1447,1555,1692,1693,1720,2167,2229
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 RewardsFee
uint256 deadFees
uint256 marketingFees
uint256 liquidityFee
uint256 devFees
uint256 burntokens
uint256 dividendsFromBuy
uint256 dividendsFromSell
uint256 feePortions
bool success
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	TRITON.sol#L784,1743,1848
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.

```
string memory _name
string memory _symbol
uint256[] memory _balances
```

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	TRITON.sol#L480,1853,2154
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.

```
_owner = msgSender
defaultToken = token
```

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	TRITON.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 ^0.8.19;
```

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	TRITON.sol#L1776,2196
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.

```
this.transferFrom(msg.sender, contributor, _balances[j])
TRITONContract.transfer(account, received)
```

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		
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		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-

	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	distributeDividends	External	Payable	-
	withdrawDividend	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
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			

	toInt256Safe	Internal		
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	✓	-
DividendPayingToken	Implementation	ERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
		External	Payable	-
	distributeDividends	Public	Payable	-
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

	_setBalance	Internal	✓	
TRITON	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	decimals	Public		-
		External	Payable	-
	updateStakingAmounts	Public	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	setPresaleWallet	External	✓	onlyOwner
	setExcludeFees	Public	✓	onlyOwner
	setExcludeDividends	Public	✓	onlyOwner
	setIncludeDividends	Public	✓	onlyOwner
	setCanTransferBefore	External	✓	onlyOwner
	setLimitsInEffect	External	✓	onlyOwner
	setGasPriceLimit	External	✓	onlyOwner
	setcooldowntimer	External	✓	onlyOwner
	setmaxWallet	External	✓	onlyOwner
	enableStaking	Public	✓	onlyOwner
	stake	Public	✓	-
	setSwapTriggerAmount	Public	✓	onlyOwner
	enableSwapAndLiquify	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	setAllowCustomTokens	Public	✓	onlyOwner

	setAllowAutoReinvest	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	transferAdmin	Public	✓	onlyOwner
	updateTransferFee	Public	✓	onlyOwner
	updateFees	Public	✓	onlyOwner
	getStakingInfo	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	setAutoClaim	External	✓	-
	setReinvest	External	✓	-
	setDividendsPaused	External	✓	onlyOwner
	isExcludedFromAutoClaim	External		-
	isReinvest	External		-
	_transfer	Internal	✓	

	getStakingBalance	Private		
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	
	updatePayoutToken	Public	✓	onlyOwner
	getPayoutToken	Public		-
	setMinimumTokenBalanceForAutoDividends	Public	✓	onlyOwner
	setMinimumTokenBalanceForDividends	Public	✓	onlyOwner
	addLiquidity	Private	✓	
	forceSwapAndSendDividends	Public	✓	onlyOwner
	swapAndSendDividends	Private	✓	
	multiSend	Public	✓	onlyOwner
	airdropToWallets	External	✓	onlyOwner
TRITONDividendTracker	Implementation	DividendPayingToken, Ownable		
		Public	✓	DividendPayingToken
	decimals	Public		-
	name	Public		-
	symbol	Public		-
	_transfer	Internal		
	withdrawDividend	Public		-
	isExcludedFromAutoClaim	External		onlyOwner
	isReinvest	External		onlyOwner

	setAllowCustomTokens	External	✓	onlyOwner
	setAllowAutoReinvest	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	includeFromDividends	External	✓	onlyOwner
	setAutoClaim	External	✓	onlyOwner
	setReinvest	External	✓	onlyOwner
	setMinimumTokenBalanceForAutoDividends	External	✓	onlyOwner
	setMinimumTokenBalanceForDividends	External	✓	onlyOwner
	setDividendsPaused	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
	updateUniswapV2Router	Public	✓	onlyOwner
	updatePayoutToken	Public	✓	onlyOwner
	getPayoutToken	Public		-
	_reinvestDividendOfUser	Private	✓	
	_withdrawDividendOfUser	Internal	✓	
IterableMapping	Library			

	get	Internal		
	getIndexOfKey	Internal		
	getKeyAtIndex	Internal		
	size	Internal		
	set	Internal	✓	
	remove	Internal	✓	

Inheritance Graph

Flow Graph

Summary

Triton 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 manipulate the fees. 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