Cyberscope

# Audit Report

# VOLTRON

May 2023

SHA256 616f734630a7e9bdbaef758042b0c3a9fedb6963a4af81c1472f2252493943f3

Audited by © cyberscope

# Table of Contents

**Table of Contents** 1
**Review** 3
Audit Updates 3
Source Files 3
**Findings Breakdown** 4
**Analysis** 5
ST - Stops Transactions 6
Description 6
Recommendation 6
BC - Blacklists Addresses 7
Description 7
Recommendation 7
**Diagnostics** 8
RVA - Redundant Variable Assignment 10
Description 10
Recommendation 10
MEM - Misleading Error Messages 11
Description 11
Recommendation 11
DDP - Decimal Division Precision 12
Description 12
Recommendation 12
PTRP - Potential Transfer Revert Propagation 13
Description 13
Recommendation 13
UFA - Unused Function Argument 14
Description 14
Recommendation 14
PVC - Price Volatility Concern 15
Description 15
Recommendation 15
RBF - Redundant BuyBack Functionality 16
Description 16
Recommendation 16
ZD - Zero Division 17
Description 17
Recommendation 17
RMAM - Redundant Modifier And Mapping 18
Description 18

Recommendation 18
RDM - Require Descriptive Message 19
Description 19
Recommendation 19
MMN - Misleading Method Naming 20
Description 20
Recommendation 20
RED - Redudant Event Declaration 21
Description 21
Recommendation 21
GO - Gas Optimization 22
Description 22
Recommendation 22
RSML - Redundant SafeMath Library 23
Description 23
Recommendation 23
IDI - Immutable Declaration Improvement 24
Description 24
Recommendation 24
L02 - State Variables could be Declared Constant 25
Description 25
Recommendation 25
L04 - Conformance to Solidity Naming Conventions 26
Description 26
Recommendation 27
L05 - Unused State Variable 28
Description 28
Recommendation 28
L07 - Missing Events Arithmetic 29
Description 29
Recommendation 29
L09 - Dead Code Elimination 30
Description 30
Recommendation 30
L11 - Unnecessary Boolean equality 31
Description 31
Recommendation 31
L16 - Validate Variable Setters 32
Description 32
Recommendation 32
L19 - Stable Compiler Version 33
Description 33

Recommendation 33
**Functions Analysis** **34**
**Inheritance Graph** **39**
**Flow Graph** **40**
**Summary** **41**
**Disclaimer** **42**
**About Cyberscope** **43**

# Review

| | |
|---|---|
| **Testing Deploy** | https://testnet.bscscan.com/address/0xa5b001b4ab5b522e59f122b7406be879dc4edc64 |
| **Address** | 0xa5b001b4ab5b522e59f122b7406be879dc4edc64 |

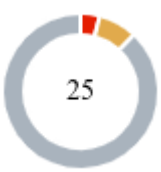
## Audit Updates

| | |
|---|---|
| **Initial Audit** | 14 May 2023 |

## Source Files

| | |
|---|---|
| **Filename** | SHA256 |
| **contracts/voltron.sol** | 616f734630a7e9bdbaef758042b0c3a9fedb6963a4af81c1472f2252493943f3 |

# Findings Breakdown




| | |
|---|---|
| Critical | 1 |
| Medium | 2 |
| Minor / Informative | 22 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 1 | 0 | 0 | 0 |
| Medium | 2 | 0 | 0 | 0 |
| Minor / Informative | 22 | 0 | 0 | 0 |

# Analysis

Critical Medium Minor / Informative Pass

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

## ST - Stops Transactions

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | contracts/voltron.sol#L372,539 |
| **Status** | Unresolved |

### Description

The contract authorized users have the authority to stop the transactions for all users excluding the users that are excluded from fees. The authorized users may take advantage of it by setting the `_maxTxAmount` to zero.

```
function setTxLimit(uint256 amount) external authorized {
    // require(amount >= _totalSupply / 1000);
    _maxTxAmount = amount;
}
```

Initially, the contract does not allow the non-excluded addresses to transfer tokens. The restriction can be resumed once an authorized user enables them.

```
if(!isTxLimitExempt[sender] && !isTxLimitExempt[recipient]){
    require(start == true,"Trading not started yet");
}
```

### Recommendation

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

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

## BC - Blacklists Addresses

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | contracts/voltron.sol#L544 |
| **Status** | Unresolved |

### Description

The contract authorized users have the authority to stop addresses from transactions. The authorized users may take advantage of it by calling the `setBlacklisted` function.

```
function setBlacklisted(address account, bool value) external authorized {
    blackListed[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. Some suggestions are:

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

# Diagnostics

Critical Medium Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | RVA | Redundant Variable Assignment | Unresolved |
| ● | MEM | Misleading Error Messages | Unresolved |
| ● | DDP | Decimal Division Precision | Unresolved |
| ● | PTRP | Potential Transfer Revert Propagation | Unresolved |
| ● | UFA | Unused Function Argument | Unresolved |
| ● | PVC | Price Volatility Concern | Unresolved |
| ● | RBF | Redundant BuyBack Functionality | Unresolved |
| ● | ZD | Zero Division | Unresolved |
| ● | RMAM | Redundant Modifier And Mapping | Unresolved |
| ● | RDM | Require Descriptive Message | Unresolved |
| ● | MMN | Misleading Method Naming | Unresolved |
| ● | RED | Redudant Event Declaration | Unresolved |
| ● | GO | Gas Optimization | Unresolved |
| ● | RSML | Redundant SafeMath Library | Unresolved |

| | | | |
|---|---|---|---|
| ● | IDI | Immutable Declaration Improvement | 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 |
| ● | L11 | Unnecessary Boolean equality | Unresolved |
| ● | L16 | Validate Variable Setters | Unresolved |
| ● | L19 | Stable Compiler Version | Unresolved |

## RVA - Redundant Variable Assignment

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L254 |
| **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 assigns the dead address in two different variables. As a result, the same value exists twice.

```
address DEAD = 0x000000000000000000000000000000000000dEaD;
address DEAD_NON_CHECKSUM = 0x000000000000000000000000000000000000dEaD;
```

### 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.

The contract should keep only one of the two variables.

## MEM - Misleading Error Messages

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L370 |
| **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(blackListed[sender]== false &&
blackListed[recipient]==false,"account" );
```

### Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use more specific and descriptive error messages.

# DDP - Decimal Division Precision

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L459 |
| **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 amountBNBLiquidity =
amountBNB.mul(dynamicLiquidityFee).div(totalBNBFee).div(2);
uint256 amountBNBMarketing = amountBNB.mul(marketingFee).div(totalBNBFee);
uint256 amountBNBDeveloper = amountBNB.mul(developerFee).div(totalBNBFee);
```

## Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

# PTRP - Potential Transfer Revert Propagation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L463 |
| **Status** | Unresolved |

## Description

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

```
payable(marketingFeeReceiver).transfer(amountBNBMarketing);
payable(developerFeeReceiver).transfer(amountBNBDeveloper);
```

## 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.

## UFA - Unused Function Argument

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L411 |
| **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 function `getTotalFee` declares a boolean argument with no name that is not being used in the function. As a result, the argument is redundant.

```
function getTotalFee(bool )
```

### 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.

The team is advised to remove the redundant argument.

# PVC - Price Volatility Concern

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L585 |
| **Status** | Unresolved |

## Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` 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 setSwapBackSettings(bool _enabled, uint256 _amount) external
authorized {
    swapEnabled = _enabled;
    swapThreshold = _amount;
}
```

## Recommendation

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

## RBF - Redundant BuyBack Functionality

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L499 |
| **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 buyback functionality. The `autoBuybackAmount`, which is the amount of tokens that is burned, is always zero. As a result, this functionality is redundant.

```
function triggerAutoBuyback() internal {
    buyTokens(autoBuybackAmount, DEAD);
    autoBuybackBlockLast = block.number;
    autoBuybackAccumulator =
autoBuybackAccumulator.add(autoBuybackAmount);
    if(autoBuybackAccumulator > autoBuybackCap){ autoBuybackEnabled =
false; }
}
```

### 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.

The team is advised to either remove the buyback functionality or set a value that is greater than zero to the `autoBuybackAmount` variable.

# ZD - Zero Division

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | contracts/voltron.sol#L439 |
| **Status** | Unresolved |

## Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
uint256 amountToLiquify =
swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
```

## Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

## RMAM - Redundant Modifier And Mapping

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L297,341 |
| **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 declares the `buyBacker` mapping along with a `onlyBuybacker` modifier. The modifier is not being used by the contract. As a result, both the mapping and the modifier are redundant.

```
mapping (address => bool) buyBacker;
modifier onlyBuybacker() { require(buyBacker[msg.sender] == true, ""); _; }
```

### 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.

The team is advised to remove these segments from the contract.

# RDM - Require Descriptive Message

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L341,576 |
| **Status** | Unresolved |

## Description

The `require()` function is used to check a condition, halt the execution of a contract and revert any changes made to the contract's state if that condition is not met. The contract does not provide a descriptive message to the `require()` function.

```
require(buyBacker[msg.sender] == true, "");
require(totalFee < feeDenominator/4);
```

## Recommendation

The team is suggested to provide a descriptive message to the `require()` function. 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.

## MMN - Misleading Method Naming

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L492 |
| **Status** | Unresolved |

### Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The `SolarFlare` method is used to manually trigger the buyback functionality, but its name does not represent its purpose.

```
function SolarFlare(uint256 amount) external authorized {
    buyTokens(amount, DEAD);

}
```

### Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

# RED - Redudant Event Declaration

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L608 |
| **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 event `BuybackMultiplierActive` is declared and not being used in the contract. As a result, it is redundant.

```
event BuybackMultiplierActive(uint256 duration);
```

## 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.

# GO - Gas Optimization

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L585 |
| **Status** | Unresolved |

## Description

Gas optimization refers to the process of reducing the amount of gas required to execute a transaction. Gas is the unit of measurement used to calculate the fees paid to miners for including a transaction in a block on the blockchain.

The contract variable `swapThreshold` is used to set a threshold where the contract will trigger the swap functionality. Since there is no input validation at the `setSwapBackSettings` function, the `swapThreshold` could be set to any amount such as zero. As a result, the contract will execute the swap functionality every time a transaction is taking place and consume more gas.

```
function setSwapBackSettings(bool _enabled, uint256 _amount) external authorized {
    swapEnabled = _enabled;
    swapThreshold = _amount;
}
```

## Recommendation

The contract could embody a check for not allowing setting the swapThreshold less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. The team 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.

## RSML - Redundant SafeMath Library

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol |
| **Status** | Unresolved |

### Description

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

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

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

### Recommendation

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

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

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

## IDI - Immutable Declaration Improvement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L315,316,318 |
| **Status** | Unresolved |

### Description

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

```
router
pair
WBNB
```

### 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.

## L02 - State Variables could be Declared Constant

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L252,253,254,260,300,305 |
| **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
address ZERO = 0x0000000000000000000000000000000000000000
address DEAD_NON_CHECKSUM = 0x000000000000000000000000000000000000dEaD
uint256 _totalSupply = 100000*10**9 * (10 ** _decimals)
uint256 autoBuybackAmount
uint256 distributorGas = 300000
```

### Recommendation

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

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L200,251,252,253,254,256,257,258,260,261,263,264,492,519,569,579,585,590 |
| **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 public WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
address DEAD_NON_CHECKSUM = 0x000000000000000000000000000000000000dEaD
string constant _name = "Votron Token"
string constant _symbol = "VOLTN"
uint8 constant _decimals = 18
uint256 _totalSupply = 100000*10**9 * (10 ** _decimals)
uint256 public _maxTxAmount = _totalSupply.div(400)
mapping (address => uint256) _balances
mapping (address => mapping (address => uint256)) _allowances

...
```

## 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** | contracts/voltron.sol#L254,305 |
| **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.

```
address DEAD_NON_CHECKSUM = 0x000000000000000000000000000000000000dEaD
uint256 distributorGas = 300000
```

### 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** | contracts/voltron.sol#L521,541,570,587,591 |
| **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.

```
autoBuybackCap = _cap
_maxTxAmount = amount
liquidityFee = _liquidityFee
swapThreshold = _amount
targetLiquidity = _target
```

### Recommendation

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

# L09 - Dead Code Elimination

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L529 |
| **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 launched() internal view returns (bool) {
        return launchedAt != 0;
    }
```

## 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.

## L11 - Unnecessary Boolean equality

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L341,370,373 |
| **Status** | Unresolved |

### Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(buyBacker[msg.sender] == true, "")
require(blackListed[sender]== false &&
blackListed[recipient]==false,"account" )
require(start == true,"Trading not started yet")
```

### Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L186,580,581,582 |
| **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 = adr
autoLiquidityReceiver = _autoLiquidityReceiver
marketingFeeReceiver = _marketingFeeReceiver
developerFeeReceiver = _developerFeeReceiver
```

### Recommendation

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

## L19 - Stable Compiler Version

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/voltron.sol#L29 |
| **Status** | Unresolved |

### Description

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

```
pragma solidity ^0.8.0;
```

### Recommendation

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

# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **SafeMath** | Library | | | |
| | tryAdd | Internal | | |
| | trySub | Internal | | |
| | tryMul | Internal | | |
| | tryDiv | Internal | | |
| | tryMod | Internal | | |
| | add | Internal | | |
| | sub | Internal | | |
| | mul | Internal | | |
| | div | Internal | | |
| | mod | Internal | | |
| | sub | Internal | | |
| | div | Internal | | |
| | mod | Internal | | |
| | | | | |
| **IBEP20** | Interface | | | |
| | totalSupply | External | | - |
| | decimals | External | | - |

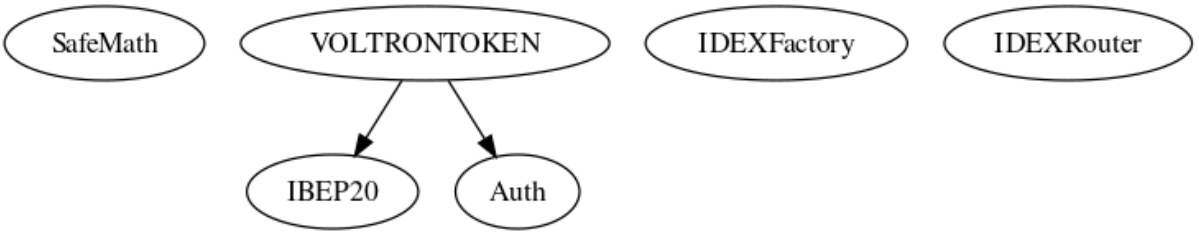
| | | | | |
|---|---|---|---|---|
| | symbol | External | | - |
| | name | External | | - |
| | getOwner | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **Auth** | Implementation | | | |
| | | Public | ✓ | - |
| | authorize | Public | ✓ | onlyOwner |
| | unauthorize | Public | ✓ | onlyOwner |
| | isOwner | Public | | - |
| | isAuthorized | Public | | - |
| | transferOwnership | Public | ✓ | onlyOwner |
| | | | | |
| **IDEXFactory** | Interface | | | |
| | createPair | External | ✓ | - |
| | | | | |
| **IDEXRouter** | Interface | | | |
| | factory | External | | - |
| | WETH | External | | - |

| | | | | |
|---|---|---|---|---|
| | addLiquidity | External | ✓ | - |
| | addLiquidityETH | External | Payable | - |
| | swapExactTokensForTokensSupportingFeeOnTransferTokens | External | ✓ | - |
| | swapExactETHForTokensSupportingFeeOnTransferTokens | External | Payable | - |
| | swapExactTokensForETHSupportingFeeOnTransferTokens | External | ✓ | - |
| | | | | |
| **VOLTRONTOKEN** | Implementation | IBEP20, Auth | | |
| | | Public | ✓ | Auth |
| | | External | Payable | - |
| | totalSupply | External | | - |
| | decimals | External | | - |
| | symbol | External | | - |
| | name | External | | - |
| | getOwner | External | | - |
| | balanceOf | Public | | - |
| | allowance | External | | - |
| | approve | Public | ✓ | - |
| | approveMax | External | ✓ | - |
| | transfer | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | _transferFrom | Internal | ✓ | |
| | _basicTransfer | Internal | ✓ | |

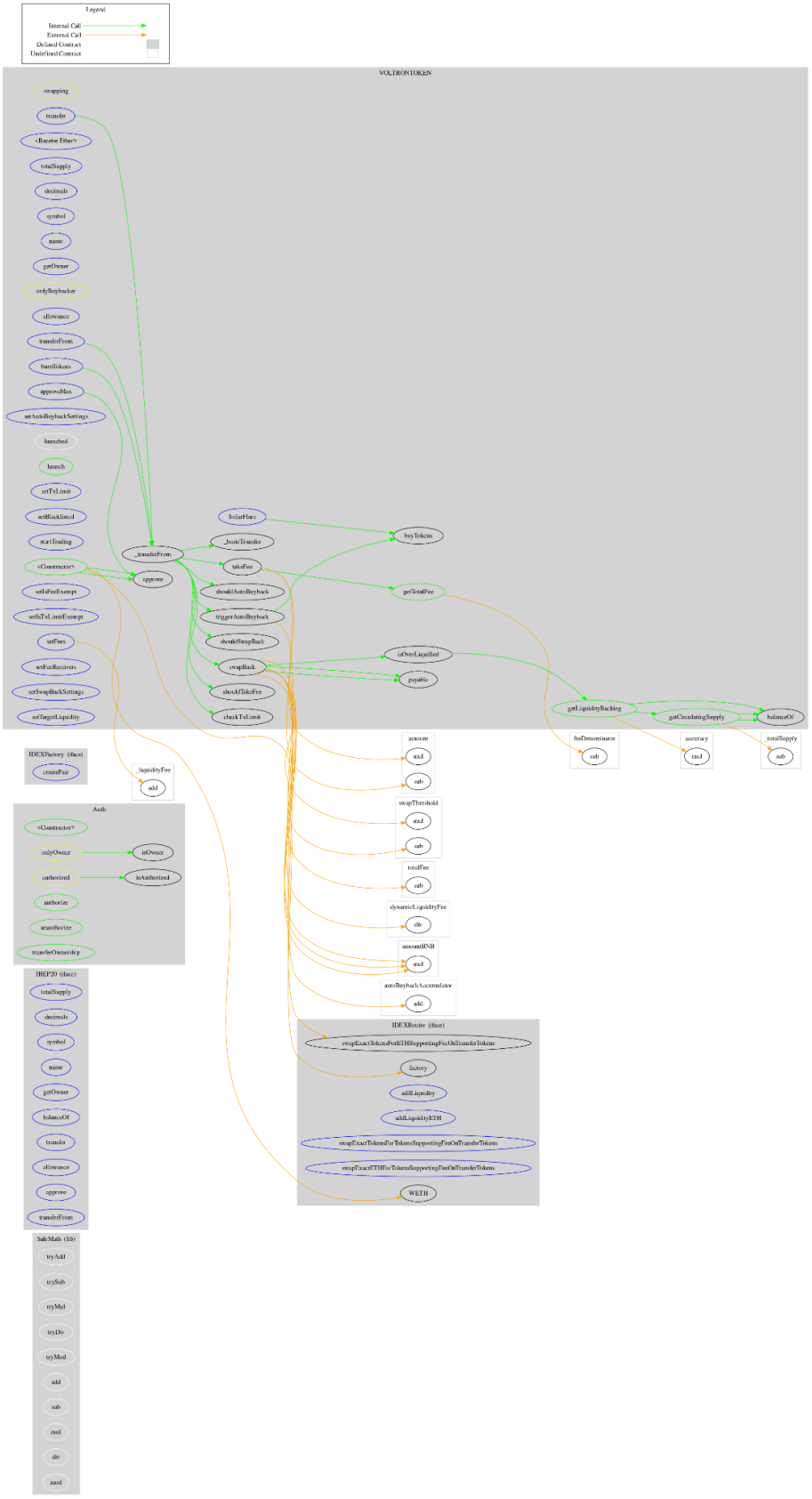
| | | | | |
|---|---|---|---|---|
| | checkTxLimit | Internal | | |
| | shouldTakeFee | Internal | | |
| | getTotalFee | Public | | - |
| | takeFee | Internal | ✓ | |
| | shouldSwapBack | Internal | | |
| | swapBack | Internal | ✓ | swapping |
| | shouldAutoBuyback | Internal | | |
| | SolarFlare | External | ✓ | authorized |
| | triggerAutoBuyback | Internal | ✓ | |
| | buyTokens | Internal | ✓ | swapping |
| | setAutoBuybackSettings | External | ✓ | authorized |
| | launched | Internal | | |
| | launch | Public | ✓ | authorized |
| | setTxLimit | External | ✓ | authorized |
| | setBlacklisted | External | ✓ | authorized |
| | startTrading | External | ✓ | authorized |
| | burnTokens | External | ✓ | authorized |
| | setIsFeeExempt | External | ✓ | authorized |
| | setIsTxLimitExempt | External | ✓ | authorized |
| | setFees | External | ✓ | authorized |
| | setFeeReceivers | External | ✓ | authorized |
| | setSwapBackSettings | External | ✓ | authorized |
| | setTargetLiquidity | External | ✓ | authorized |

| | | | | |
|---|---|---|---|---|
| | getCirculatingSupply | Public | | - |
| | getLiquidityBacking | Public | | - |
| | isOverLiquified | Public | | - |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
VOLTRONTOKEN
swapping
transfer
<Receive Ether>
totalSupply
decimals
symbol
name
getOwner
onlyBuybacker
allowance
transferFrom
burnTokens
approveMax
setAutoBuybackSettings
launched
launch
setTxLimit
setBlacklisted
startTrading
<Constructor>
setIsFeeExempt
setIsTxLimitExempt
setFees
setFeeReceivers
setSwapBackSettings
setTargetLiquidity
_transferFrom
approve
$olarHarc
_basicTransfer
takeFee
shouldAutoBuyback
triggerAutoBuyback
shouldSwapBack
swapBack
shouldTakeFee
checkTxLimit
buyTokens
getTotalFee
isOverLiquified
payable
getLiquidityBacking
getCirculatingSupply
balanceOf
feeDenominator
sub
accuracy
mul
_totalSupply
sub
amount
mul
sub
swapThreshold
mul
sub
totalFee
sub
dynamicLiquidityFee
div
amountBNB
mul
autoBuybackAccumulator
add
IDEXFactory (iface)
createPair
_liquidityFee
add
Auth
<Constructor>
onlyOwner
isOwner
authorized
isAuthorized
authorize
unauthorize
transferOwnership
IBEP20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
IDEXRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
factory
addLiquidity
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
WETH
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod

# Summary

VOLTRON 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 authorized users like stopping transactions and blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Renouncing ownership of the contract will not eliminate the contract threats, since the authorized users will still have access to the authorized functions. There is also a limit of max 24% fees, except for the first block after the contract launches, which is 99%.

# 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