Cyberscope

Audit Report
x314

April 2024

Repository	https://github.com/LongSwapLabs/smartcontract
Address	9bfc76657d6a9f21adae907e4c30ba8d5eae3195
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Overview **4**
Findings Breakdown **6**
Diagnostics **7**
BC - Blacklists Addresses 9
Description 9
Recommendation 10
ITSC - Inaccurate Token Swap Calculation 11
Description 11
Recommendation 11
IVPE - Inadequate Vesting Period Enforcement 13
Description 13
Recommendation 14
CCR - Contract Centralization Risk 15
Description 15
Recommendation 15
IDI - Immutable Declaration Improvement 16
Description 16
Recommendation 16
ITS - Inconsistent Taxation Settings 17
Description 17
Recommendation 18
ILIC - Interface Located Inside Contract 20
Description 20
Recommendation 21
MMN - Misleading Method Naming 22
Description 22
Recommendation 23
MSC - Missing Sanity Check 24
Description 24
Recommendation 24
RV - Redundant Variable 25
Description 25
Recommendation 26
TSD - Total Supply Diversion 27
Description 27

Recommendation 28
L02 - State Variables could be Declared Constant 29
Description 29
Recommendation 29
L04 - Conformance to Solidity Naming Conventions 30
Description 30
Recommendation 31
L07 - Missing Events Arithmetic 32
Description 32
Recommendation 32
L09 - Dead Code Elimination 33
Description 33
Recommendation 33
L11 - Unnecessary Boolean equality 34
Description 34
Recommendation 34
L13 - Divide before Multiply Operation 35
Description 35
Recommendation 35
L16 - Validate Variable Setters 36
Description 36
Recommendation 36
Functions Analysis 37
Inheritance Graph 39
Flow Graph 40
Summary 41
Disclaimer 42
About Cyberscope 43

Review

Repository	https://github.com/LongSwapLabs/smartcontract
Commit	9bfc76657d6a9f21adae907e4c30ba8d5eae3195

Audit Updates

Initial Audit	16 Apr 2024

Source Files

Filename	SHA256
src/ERC314.sol	1f91aa258f6358e96a9c8bd8f6cd3e74e8aaca8162337ae73082e28312f81673
openzeppelin-contracts/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81
openzeppelin-contracts/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6

Overview

This Token contract not only implements the basic ERC20 functions but also enhances the standard by incorporating features that enable the Token's contract to be utilized as liquidity, including functions such as `buy()` and `sell()`. Moreover, it has a function that may be called by any address and is decreasing the token supply by 0.25% of the current supply per hour. While there is also a `presale()` function that allows the owner to allocate the presale amount (which is 20% of the initial total supply) to the addresses given to the function. The `presale()` function can be called once.

Additionally, the contract incorporates a buy and sell tax, which is set during initialization in the constructor and can be modified by the owner. It also includes a cooling period, preventing users from transferring tokens for a specified number of blocks. Finally, there is a maximum holding amount per user, set at 0.5% of the initial total supply.

Roles

Owner

The contract's owner who is set at contract's constructor can interact with the following functions:

- presale(address[] memory _investors) public onlyOwner
- setMaxWallet(uint256 *maxWallet*) external onlyOwner
- setLastTransaction(address[] memory accounts, uint32 _block) external
- setExcludeCoolingOf(address[] memory accounts, bool _ok) external onlyOwner
- setBuyTax(uint32 _buyTax) external onlyOwner
- setSellTax(uint32 _sellTax) external onlyOwner
- setCooling(uint32 _coolingBlock) external onlyOwner
- addLiquidity(uint32 _blockToUnlockLiquidity) public payable onlyOwner

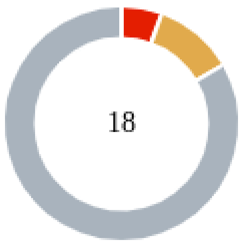
LiquidityProvider

The contract's liquidityProvider who is set at the function addLiquidity can interact with the following functions:

- removeLiquidity() public onlyLiquidityProvider

- extendLiquidityLock(uint32 _blockToUnlockLiquidity) public onlyLiquidityProvider

Findings Breakdown



●	Critical	1
●	Medium	2
●	Minor / Informative	15

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	2	0	0	0
●	Minor / Informative	15	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
🔴	BC	Blacklists Addresses	Unresolved
🟠	ITSC	Inaccurate Token Swap Calculation	Unresolved
🟠	IVPE	Inadequate Vesting Period Enforcement	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	ITS	Inconsistent Taxation Settings	Unresolved
●	ILIC	Interface Located Inside Contract	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	RV	Redundant Variable	Unresolved
●	TSD	Total Supply Diversion	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved

●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved

BC - Blacklists Addresses

Criticality	Critical
Location	ERC314.sol#L239
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `setLastTransaction` function, with the list of addresses he wants to stop from transactions and using an arbitrary high block number.

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal virtual {
    require(to != address(0), "ERC20: transfer to the zero address");

    if (!excludeCoolingOf[_msgSender()]) {
        require(
            lastTransaction[_msgSender()] + coolingBlock < block.number,
            "You can't make two transactions in the cooling block"
        );
        lastTransaction[_msgSender()] = uint32(block.number);
    }
    ...
}
...
function setLastTransaction(
    address[] memory accounts,
    uint32 _block
) external onlyOwner {
    for (uint i = 0; i < accounts.length; i++) {
        lastTransaction[accounts[i]] = _block;
    }
}
```

Recommendation

The team is advised to use limits to `_block` function parameter in reference to current block number.

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.

ITSC - Inaccurate Token Swap Calculation

Criticality	Medium
Location	ERC314.sol#L252,262,279
Status	Unresolved

Description

During the assessment of the smart contract, it was observed that the formula used to calculate the `tokenAmount` in the `buy()` function is not accurate. Specifically, the denominator in the calculation only considers the `reserveIn` balance, but it should also include the amount being swapped `msg.value` to ensure proper calculation of the `tokenAmount`. These calculations are missing consideration for the amount being swapped, leading to inaccurate token amounts.

```
function buy() internal {
    ...
    uint256 tokenAmount = (msg.value * _balances[address(this)]) /
(address(this).balance);
    ...
}

function sell(address owner, uint256 amount) internal {
    ...
    uint256 fee = (amount * sellTax) / 100;
    uint256 sellAmount = amount - fee;

    uint256 ethAmount = (sellAmount * address(this).balance) /
(_balances[address(this)] + sellAmount);
    ...
}
```

Recommendation

The team is advised to utilize the `getAmountOut()` function provided in the contract, which accurately calculates the token amount based on the swap value and reserves. This function is already implemented and can be utilized to ensure correct token amount calculations.

By utilizing the `getAmountOut()` function, the contract will accurately calculate the `tokenAmount` based on the swap value and reserves, enhancing the reliability of the contract.

IVPE - Inadequate Vesting Period Enforcement

Criticality	Medium
Location	ERC314.sol#L222,246
Status	Unresolved

Description

The functions `addLiquidity` and `extendLiquidityLock` in the smart contract do not adequately enforce the vesting period for liquidity providers. While both functions check if `_blockToUnlockLiquidity` is greater than the current block number, they fail to verify whether the specified vesting period meets certain requirements, such as being at least a week.

This oversight could potentially lead to liquidity being unlocked prematurely or not being locked for a sufficient period, which may pose risks to the project and its participants.

```
function addLiquidity(uint32 _blockToUnlockLiquidity) public payable
onlyOwner {
    require(liquidityAdded == false, "Liquidity already added");

    liquidityAdded = true;

    require(msg.value > 0, "No ETH sent");
    require(block.number < _blockToUnlockLiquidity, "Block number too
low");

    blockToUnlockLiquidity = _blockToUnlockLiquidity;
    liquidityProvider = _msgSender();

    emit AddLiquidity(_blockToUnlockLiquidity, msg.value);
}
...
function extendLiquidityLock(uint32 _blockToUnlockLiquidity) public
onlyLiquidityProvider {
    require(blockToUnlockLiquidity < _blockToUnlockLiquidity, "You can't
shorten duration");

    blockToUnlockLiquidity = _blockToUnlockLiquidity;
}
```

Recommendation

To mitigate this issue, it is recommended to implement additional checks in both functions to enforce the minimum vesting period required by the project. This could involve validating the duration against a predefined threshold, such as ensuring that the difference between `_blockToUnlockLiquidity` and the current block number corresponds to at least a week, depending on the project's requirements.

By enforcing a minimum vesting period, the smart contract can better protect the interests of liquidity providers and ensure the stability and integrity of the project's liquidity management system.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	ERC314.sol#L75,191,195,201,207,212,217
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
presale(address[] memory _investors) public onlyOwner
setMaxWallet(uint256 _maxWallet_) external onlyOwner
setLastTransaction(address[] memory accounts, uint32 _block) external
onlyOwner
setExcludeCoolingOf(address[] memory accounts, bool _ok) external
onlyOwner
setBuyTax(uint32 _buyTax) external onlyOwner
setSellTax(uint32 _sellTax) external onlyOwner
setCooling(uint32 _coolingBlock) external onlyOwner
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ERC314.sol#L72,82
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.

```
_totalSupply
presaleAmount
```

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.

ITS - Inconsistent Taxation Settings

Criticality	Minor / Informative
Location	ERC314.sol#L62,257,262
Status	Unresolved

Description

The contract implements taxes on buy and sell transactions. However, there is a discrepancy in the implementation of tax restrictions between the constructor and the external functions `setBuyTax()` and `setSellTax()`.

Although the contract includes functions `setBuyTax()` and `setSellTax()` to modify these tax rates after deployment, they enforce a restriction that the tax rates cannot exceed 50%. However, this constraint is not applied during the contract initialization, leading to a potential inconsistency between the initial tax rates and the maximum allowable tax rate.

```
constructor(
    string memory name_,
    string memory symbol_,
    uint256 totalSupply_,
    uint32 _coolingBlock,
    uint32 _sellTax,
    uint32 _buyTax
) {
    _name = name_;
    _symbol = symbol_;
    _totalSupply = totalSupply_;

    _maxWallet = totalSupply_ / 200;

    coolingBlock = _coolingBlock;
    sellTax = _sellTax;
    buyTax = _buyTax;

    _lastRebaseTime = block.timestamp + 1 days;

    presaleAmount = (totalSupply_ * 2) / 10;

    uint256 liquidityAmount = totalSupply_ - presaleAmount;
    _balances[address(this)] = liquidityAmount;
}
...
function setBuyTax(uint32 _buyTax) external onlyOwner {
    require(_buyTax <= 50, "Tax is too big");
    buyTax = _buyTax;
}

function setSellTax(uint32 _sellTax) external onlyOwner {
    require(_sellTax <= 50, "Tax is too big");
    sellTax = _sellTax;
}
```

Recommendation

To ensure the contract's tax rates remain within the intended bounds and maintain consistent behavior, it's important to enforce a maximum tax limit of 50% during both contract initialization and subsequent modifications using the `setBuyTax()` and `setSellTax()` functions. By implementing this constraint uniformly across all tax-related functionalities, such as deployment and adjustments, the contract can mitigate the risk of discrepancies and unexpected behavior, enhancing the reliability and predictability of its taxation mechanism.

ILIC - Interface Located Inside Contract

Criticality	Minor / Informative
Location	ERC314.sol#L7
Status	Unresolved

Description

The contract `ERC314` includes the interface `IEERC314` within the same file instead of importing it from an external source. While this practice may seem convenient, it introduces certain limitations that warrant attention.

Modularity and Reusability: By embedding the interface within the same file as the contract implementation, the code loses modularity. Interfaces are intended to define a set of functions that other contracts can interact with, and they often serve as blueprints for standard functionality. Separating the interface into its own file allows for easier reuse across multiple contracts, enhancing code maintainability and reducing redundancy.

Readability and Maintainability: locating the interface within the contract implementation can impact code readability and maintainability, especially in larger projects with numerous contracts and interfaces.

```
interface IEERC314 {
    event Transfer(address indexed from, address indexed to, uint256
value);
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event AddLiquidity(uint32 _blockToUnlockLiquidity, uint256 value);
    event RemoveLiquidity(uint256 value);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out
    );
}
```

Recommendation

To address the aforementioned concerns the team is advised to refactor the contract to separate the interface declaration into its own file. This action promotes modularity and enhances code readability.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	ERC314.sol#L367
Status	Unresolved

Description

The smart contract under assessment features a function labeled `rebase()` , suggesting its intended purpose to adjust the token supply in accordance with a rebase mechanism. However, upon closer examination of the code, it becomes evident that the function does not adhere to the expected behavior of a rebase token, which typically involves inflating the token supply.

Instead, the `rebase()` function in the provided code implements a burning mechanism, wherein tokens are permanently removed from circulation. This divergence from the anticipated functionality poses a significant discrepancy and potential risk to users and stakeholders who may rely on the contract's adherence to standard rebase token behavior.

This implementation deviates from the typical behavior of a rebase mechanism, which is intended to adjust the token supply in response to changing market conditions, typically resulting in an increase in the total token supply. Instead, the function effectively decreases the token supply by burning a portion of the tokens.

```
function rebase() external {
    uint256 lastRebaseTime = _lastRebaseTime;
    if (0 == lastRebaseTime) {
        return;
    }

    uint256 nowTime = block.timestamp;
    if (nowTime < lastRebaseTime + _rebaseDuration) {
        return;
    }

    _lastRebaseTime = nowTime;

    uint256 poolBalance = _balances[address(this)];
    uint256 rebaseAmount = (((poolBalance * _rebaseRate) / 10000) *
        (nowTime - lastRebaseTime)) / _rebaseDuration;

    if (rebaseAmount > poolBalance / 2) {
        rebaseAmount = poolBalance / 2;
    }

    if (rebaseAmount > 0) {
        _basicTransfer(address(this), address(0xdead), rebaseAmount);
    }
}
```

Recommendation

To ensure clarity and alignment with expectations, it's advisable to clearly define the intended functionality of the `rebase()` function. If the intention is indeed to implement a rebase mechanism, the function logic should be revised to adjust the token supply accordingly by minting or distributing tokens in response to market conditions.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	ERC314.sol#L155,252
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

```
address from
uint256 value
uint256 reserveETH
uint256 reserveToken
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

RV - Redundant Variable

Criticality	Minor / Informative
Location	ERC314.sol#L222,236
Status	Unresolved

Description

Within the contract,there is a variable named `liquidityProvider` , which is intended to store the address of the liquidity provider. This variable is primarily utilized in the `removeLiquidity` function to restrict access, ensuring that only the liquidity provider can execute it.

However, it becomes apparent that the `liquidityProvider` variable serves no practical purpose. This is due to the fact that the `addLiquidity` function, responsible for setting the value of `liquidityProvider` , is restricted by the `onlyOwner` modifier. Consequently, only the contract owner can invoke `addLiquidity` , resulting in the `liquidityProvider` variable always being assigned the owner's address.

As a result, the variable `liquidityProvider` becomes redundant, as it merely duplicates information already inherent in the ownership of the contract. Its presence introduces unnecessary complexity and does not contribute to the security or functionality of the contract.

```
   function addLiquidity(uint32 _blockToUnlockLiquidity) public payable
onlyOwner {
        require(liquidityAdded == false, "Liquidity already added");

        liquidityAdded = true;

        require(msg.value > 0, "No ETH sent");
        require(block.number < _blockToUnlockLiquidity, "Block number
too low");

        blockToUnlockLiquidity = _blockToUnlockLiquidity;
        liquidityProvider = _msgSender();

        emit AddLiquidity(_blockToUnlockLiquidity, msg.value);
    }

    function removeLiquidity() public onlyLiquidityProvider {
        require(block.number > blockToUnlockLiquidity, "Liquidity
locked");

        liquidityAdded = false;

        payable(liquidityProvider).transfer(address(this).balance);

        emit RemoveLiquidity(address(this).balance);
    }
```

Recommendation

To streamline the contract's logic and improve readability, consider removing the `liquidityProvider` variable altogether. Since only the owner can provide liquidity, there is no need to store the liquidity provider's address separately.

By removing redundant variables and simplifying the contract's structure, the contract can become more efficient and maintainable.

TSD - Total Supply Diversion

Criticality	Minor / Informative
Location	ERC314.sol#L62
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.

```
constructor(
    string memory name_,
    string memory symbol_,
    uint256 totalSupply_,
    uint32 _coolingBlock,
    uint32 _sellTax,
    uint32 _buyTax
) {
    _name = name_;
    _symbol = symbol_;
    _totalSupply = totalSupply_;

    _maxWallet = totalSupply_ / 200;

    coolingBlock = _coolingBlock;
    sellTax = _sellTax;
    buyTax = _buyTax;

    _lastRebaseTime = block.timestamp + 1 days;

    presaleAmount = (totalSupply_ * 2) / 10;

    uint256 liquidityAmount = totalSupply_ - presaleAmount;
    _balances[address(this)] = liquidityAmount;
}
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ERC314.sol#L46
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.

```
uint256 public _rebaseRate = 25
```

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	ERC314.sol#L30,46,47,88,235,241,250,257,262,267,273,299,311
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.

```
uint256 public _maxWallet
uint256 public _rebaseRate = 25
uint256 public _lastRebaseTime
address[] memory _investors
uint256 _maxWallet_
uint32 _block
bool _ok
uint32 _buyTax
uint32 _sellTax
uint32 _coolingBlock
uint32 _blockToUnlockLiquidity
bool _buy
```

Recommendation

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

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	ERC314.sol#L191,195,201,207,212,217,246
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.

```
setMaxWallet(uint256 _maxWallet_) external onlyOwner
setLastTransaction(address[] memory accounts, uint32 _block) external
onlyOwner
setExcludeCoolingOf(address[] memory accounts, bool _ok) external
onlyOwner
setBuyTax(uint32 _buyTax) external onlyOwner
setSellTax(uint32 _sellTax) external onlyOwner
setCooling(uint32 _coolingBlock) external onlyOwner
extendLiquidityLock(uint32 _blockToUnlockLiquidity) public
onlyLiquidityProvider
```

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	ERC314.sol#L369
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.

```
if (0 == lastRebaseTime) {
        return;
}
```

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	ERC314.sol#L89,275
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(presaleEnable == false, "Presale already enabled")
require(liquidityAdded == false, "Liquidity already added")
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	ERC314.sol#L328,336,381
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 tokenAmount = (msg.value * _balances[address(this)]) /
              (address(this).balance)
uint256 fee = (tokenAmount * buyTax) / 100
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ERC314.sol#L49,191,195
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.

```
uint32 _coolingBlock
uint32 _sellTax
uint32 _buyTax
uint256 _maxWallet_
uint32 _block
```

Recommendation

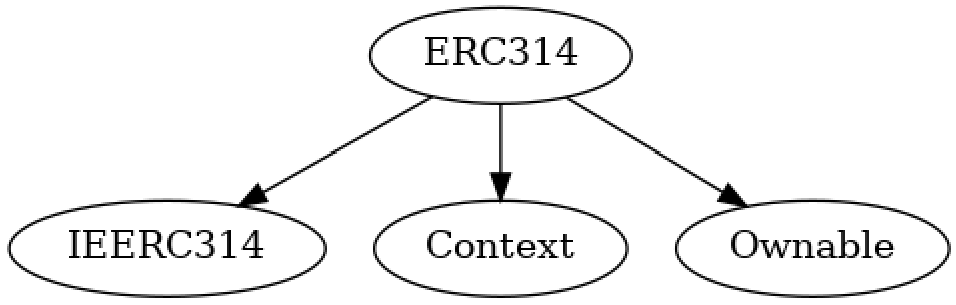
By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

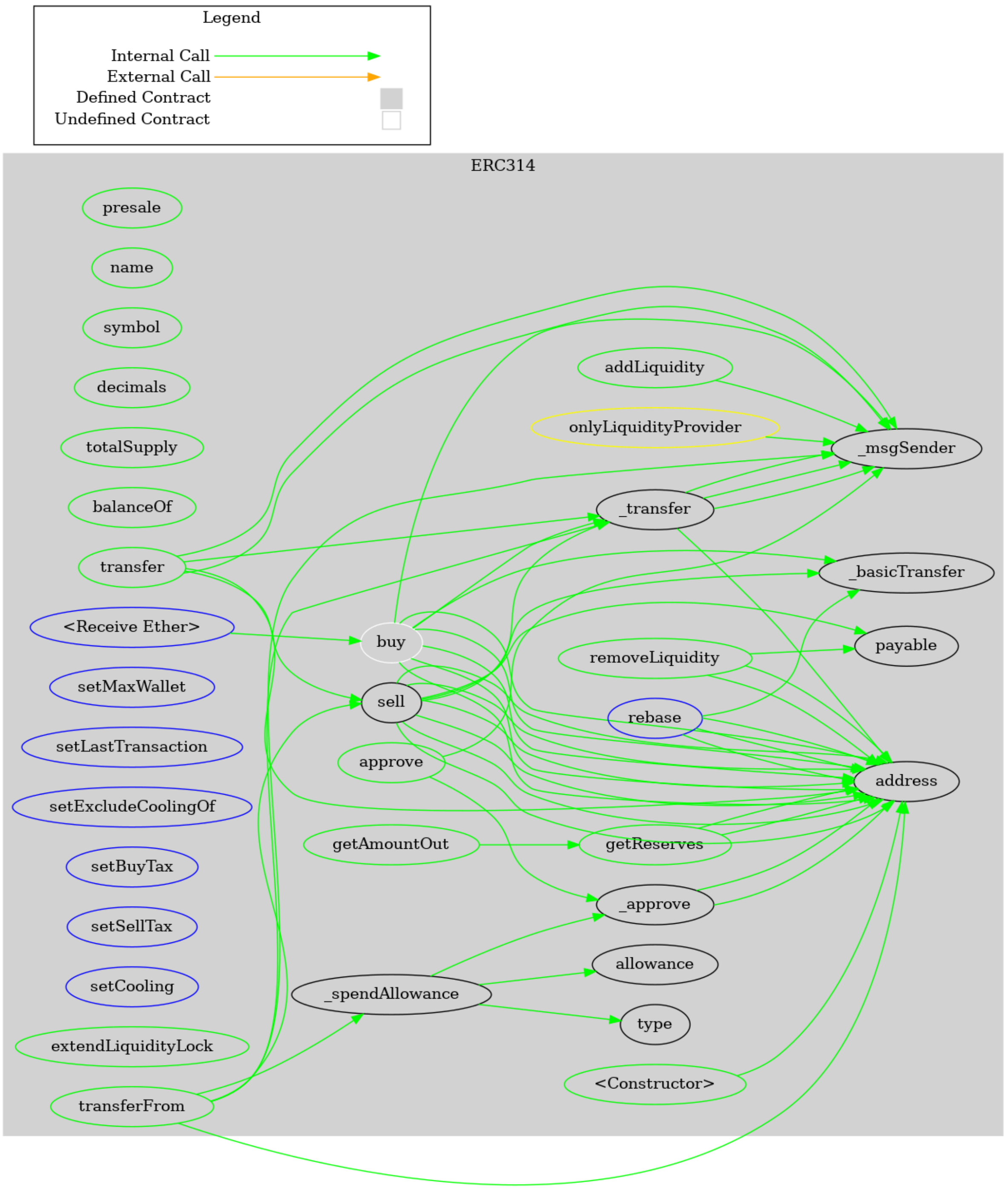
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ERC314	Implementation	Context, Ownable, IEERC314		
		Public	✓	-
	presale	Public	✓	onlyOwner
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	transfer	Public	✓	-
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_transfer	Internal	✓	
	_basicTransfer	Internal	✓	
	getReserves	Public		-

	setMaxWallet	External	✓	onlyOwner
	setLastTransaction	External	✓	onlyOwner
	setExcludeCoolingOf	External	✓	onlyOwner
	setBuyTax	External	✓	onlyOwner
	setSellTax	External	✓	onlyOwner
	setCooling	External	✓	onlyOwner
	addLiquidity	Public	Payable	onlyOwner
	removeLiquidity	Public	✓	onlyLiquidityProvider
	extendLiquidityLock	Public	✓	onlyLiquidityProvider
	getAmountOut	Public		-
	buy	Internal	✓	
	sell	Internal	✓	
	rebase	External	✓	-
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC314
presale
name
symbol
decimals
totalSupply
balanceOf
transfer
<Receive Ether>
setMaxWallet
setLastTransaction
setExcludeCoolingOf
setBuyTax
setSellTax
setCooling
extendLiquidityLock
transferFrom
addLiquidity
onlyLiquidityProvider
_msgSender
_transfer
_basicTransfer
buy
sell
approve
payable
removeLiquidity
rebase
address
getAmountOut
getReserves
_approve
allowance
_spendAllowance
type
<Constructor>

Summary

ERC314 contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

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

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

About Cyberscope

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

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

The Cyberscope team

https://www.cyberscope.io