Cyberscope

Audit Report

INBUVE

May 2025

Network ARBITRUM

Address 0xc571a6d44f63d389a4be5ed4e17abb288187f353

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	IFDL	Incorrect Fee Deduction Logic	Unresolved
Minor	CR	Code Repetition	Unresolved
Minor	CCR	Contract Centralization Risk	Unresolved
Minor	IDI	Immutable Declaration Improvement	Unresolved
Minor	ITI	Inefficient Tax Implementation	Unresolved
Minor	MC	Missing Check	Unresolved
Minor	MEE	Missing Events Emission	Unresolved
Minor	RF	Redundant Functionality	Unresolved
Minor	RSML	Redundant SafeMath Library	Unresolved
Minor	RSRS	Redundant SafeMath Require Statement	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	SimpleToken
Compiler Version	v0.8.26+commit.8a97fa7a
Optimization	200 runs
Explorer	https://arbiscan.io/address/0xc571a6d44f63d389a4be5ed4e17abb288187f353
Address	0xc571a6d44f63d389a4be5ed4e17abb288187f353
Network	ARBITRUM
Symbol	IBV
Decimals	18
Total Supply	1.000.000.000
Badge Eligibility	Yes

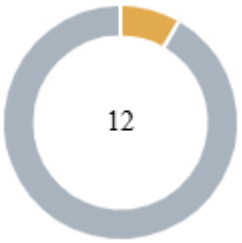
Audit Updates

Initial Audit	29 Apr 2025 https://github.com/cyberscope-io/audits/blob/main/ibv/v1/audit.pdf
Corrected Phase 2	08 May 2025

Source Files

Filename	SHA256
SimpleToken.sol	dffe925f64d4d678d6ce65819764b07ea67c3070f6a61e046e321a1075f0fd6a

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

IFDL - Incorrect Fee Deduction Logic

Criticality	Medium
Location	SimpleToken.sol#L381
Status	Unresolved

Description

The contract under review includes a tax mechanism that imposes fees on transactions based on whether the transaction is a buy, sell, or transfer. However, the current implementation contains logical errors that cause incorrect fee deductions. Specifically, the contract incorrectly deducts transfer fees in scenarios where buy or sell fees should be applied but the fees are 0, leading to potential inconsistencies in the tax amounts applied to different transactions.

- Buy Fees: If the sender is the liquidity pair and there are buy fees defined, the contract correctly deducts buy fees.
- Sell Fees: If the recipient is the liquidity pair and there are sell fees defined, the contract correctly deducts sell fees.
- Transfer Fees: If neither condition is met, but transfer fees are defined, the contract deducts transfer fees.

Case 1 Buy: `addressesLiquidity[sender]=true` and `SwapBlock.getPercentsTaxBuy().length=0` then if `SwapBlock.getPercentsTaxTransfer().length > 0` fees will be deducted.

Case 2 Sell: `addressesLiquidity[recipient]=true` and `SwapBlock.getPercentsTaxSell().length=0` then if `SwapBlock.getPercentsTaxTransfer().length > 0` fees will be deducted.

```
if (addressesLiquidity[sender] &&
SwapBlock.getPercentsTaxBuy().length > 0) {
    //...
} else if (addressesLiquidity[recipient] &&
SwapBlock.getPercentsTaxSell().length > 0) {
    //...
} else if (SwapBlock.getPercentsTaxTransfer().length > 0) {
    //...
} else {
    //...
}
```

Recommendation

Refactor the fee deduction logic. Update the conditions to ensure that transfer fees are only applied when both the sender and recipient are not liquidity pairs. By addressing these issues, the contract will correctly apply fees according to the intended logic for buy, sell, and transfer transactions, ensuring consistent and expected behavior.

CR - Code Repetition

Criticality	Minor / Informative
Location	SimpleToken.sol#L210,220,230,357,369,381
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
function setTaxBuy(uint256[] memory _percentsTaxBuy, address[]
memory _addressesTaxBuy) public onlyOwner {
    require(_percentsTaxBuy.length == _addressesTaxBuy.length,
"_percentsTaxBuy.length != _addressesTaxBuy.length");
    uint256 TaxSum = getTaxSum(_percentsTaxBuy);
    require(TaxSum <= 1, "TaxSum > 1"); // Set the maximum tax
limit
    percentsTaxBuy = _percentsTaxBuy;
    addressesTaxBuy = _addressesTaxBuy;
}
function setTaxSell(uint256[] memory _percentsTaxSell,
address[] memory _addressesTaxSell) public onlyOwner { /*almost
identical*/}
function setTaxTransfer(uint256[] memory _percentsTaxTransfer,
address[] memory _addressesTaxTransfer) public onlyOwner {
/*almost identical*/}
```

```
function _transfer(address sender, address recipient, uint256
amount) internal {
//...
    if (addressesLiquidity[sender] &&
SwapBlock.getPercentsTaxBuy().length > 0) {
        for (uint i; i < SwapBlock.getPercentsTaxBuy().length;
i++) {
            amountTax =
amount.div(100).mul(SwapBlock.getPercentsTaxBuy()[i]);
            amountRecipient = amountRecipient.sub(amountTax);
            _balances[SwapBlock.getAddressesTaxBuy()[i]] =
SafeMath.add(_balances[SwapBlock.getAddressesTaxBuy()[i]],
amountTax);
            emit Transfer(sender,
SwapBlock.getAddressesTaxBuy()[i], amountTax);
        }
        _balances[recipient] =
_balances[recipient].add(amountRecipient);
        emit Transfer(sender, recipient, amountRecipient);
    } else if (addressesLiquidity[recipient] &&
SwapBlock.getPercentsTaxSell().length > 0) {
        // almost identical
    } else if (SwapBlock.getPercentsTaxTransfer().length > 0) {
        // almost identical
    }
//...
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	SimpleToken.sol#L190,194,202,206,210,220,230
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.

```
function addAddressLiquidity(address _addressLiquidity) public
onlyOwner
function removeAddressLiquidity (address _addressLiquidity)
public onlyOwner
function addAddressIgnoreTax(address _addressIgnoreTax) public
onlyOwner
function removeAddressIgnoreTax (address _addressIgnoreTax)
public onlyOwner
function setTaxBuy(uint256[] memory _percentsTaxBuy, address[]
memory _addressesTaxBuy) public onlyOwner
function setTaxSell(uint256[] memory _percentsTaxSell,
address[] memory _addressesTaxSell) public onlyOwner
function setTaxTransfer(uint256[] memory _percentsTaxTransfer,
address[] memory _addressesTaxTransfer) public 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	SimpleToken.sol#L280,281
Status	Unresolved

Description

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

```
_decimals
_totalSupply
```

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.

ITI - Inefficient Tax Implementation

Criticality	Minor / Informative
Location	SimpleToken.sol#L154,210,220,230
Status	Unresolved

Description

The contract under review is an ERC20 token with tax mechanisms for buy, sell, and transfer operations. The taxes are defined using three arrays: `percentsTaxBuy`, `percentsTaxSell`, and `percentsTaxTransfer`. Each array holds the percentage taxes for the respective operation, and the sum of each array is restricted to be at most 1. The current implementation involves setting these tax percentages through the functions `setTaxBuy`, `setTaxSell`, and `setTaxTransfer`, which require passing arrays of percentages and corresponding addresses.

The arrays for the tax percentages are redundant due to the restriction that their sum can only be at most 1. This implies that only one element in each array can be set to 1, and the rest must be 0. Using arrays in this context introduces unnecessary complexity and gas costs.

```
function getTaxSum(uint256[] memory _percentsTax) internal pure
returns (uint256) {
    uint256 TaxSum = 0;
    for (uint i; i < _percentsTax.length; i++) {
        TaxSum = TaxSum.add(_percentsTax[i]);
    }
    return TaxSum;
}
```

```
function setTaxBuy(uint256[] memory _percentsTaxBuy, address[]
memory _addressesTaxBuy) public onlyOwner {
    //...
    uint256 TaxSum = getTaxSum(_percentsTaxBuy);
    require(TaxSum <= 1, "TaxSum > 1"); // Set the maximum tax
limit
    percentsTaxBuy = _percentsTaxBuy;
    addressesTaxBuy = _addressesTaxBuy;
}

function setTaxSell(uint256[] memory _percentsTaxSell,
address[] memory _addressesTaxSell) public onlyOwner {
    //...
    uint256 TaxSum = getTaxSum(_percentsTaxSell);
    require(TaxSum <= 1, "TaxSum > 1"); // Set the maximum tax
limit
    percentsTaxSell = _percentsTaxSell;
    addressesTaxSell = _addressesTaxSell;
}

function setTaxTransfer(uint256[] memory _percentsTaxTransfer,
address[] memory _addressesTaxTransfer) public onlyOwner {
    //...
    uint256 TaxSum = getTaxSum(_percentsTaxTransfer);
    require(TaxSum <= 1, "TaxSum > 1"); // Set the maximum tax
limit
    percentsTaxTransfer = _percentsTaxTransfer;
    addressesTaxTransfer = _addressesTaxTransfer;
}
```

Recommendation

To optimize the contract, it is recommended to replace the `uint256[]` tax arrays with a simpler `bool` type that indicates whether a tax is applied or not. This reduces the complexity and cost associated with handling arrays and ensures clarity in the contract's logic.

MC - Missing Check

Criticality	Minor / Informative
Location	SimpleToken.sol#L190,194,202,206,210,220,230
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.

`addAddressLiquidity` and `removeAddressLiquidity` are missing checks to ensure that the mappings are not already in the desired state. Additionally checks are missing to ensure that `_addressLiquidity` is not `address(0)`.

```
function addAddressLiquidity(address _addressLiquidity) public
onlyOwner {
    addressesLiquidity[_addressLiquidity] = true;
}
function removeAddressLiquidity (address _addressLiquidity)
public onlyOwner {
    addressesLiquidity[_addressLiquidity] = false;
}
```

The case is the same as above for `addAddressIgnoreTax` and `removeAddressIgnoreTax`.

```
function addAddressIgnoreTax(address _addressIgnoreTax) public
onlyOwner {
    addressesIgnoreTax[_addressIgnoreTax] = true;
}
function removeAddressIgnoreTax (address _addressIgnoreTax)
public onlyOwner {
    addressesIgnoreTax[_addressIgnoreTax] = false;
}
```

`setTaxBuy` , `setTaxSell` and `setTaxTransfer` are missing checks to ensure each `_percentsTaxTransfer` is non-zero value. Additionally a check is missing to ensure that each element in `_addressesTaxTransfer` is not `address(0)` .

```
function setTaxBuy(uint256[] memory _percentsTaxBuy, address[]
memory _addressesTaxBuy) public
function setTaxSell(uint256[] memory _percentsTaxSell,
address[] memory _addressesTaxSell) public
function setTaxTransfer(uint256[] memory _percentsTaxTransfer,
address[] memory _addressesTaxTransfer) public
```

Recommendation

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

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	SimpleToken.sol#L190,194,202,206,210,220,230
Status	Unresolved

Description

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

```
function addAddressLiquidity(address _addressLiquidity) public
onlyOwner
function removeAddressLiquidity (address _addressLiquidity)
public onlyOwner
function addAddressIgnoreTax(address _addressIgnoreTax) public
onlyOwner
function removeAddressIgnoreTax (address _addressIgnoreTax)
public onlyOwner
function setTaxBuy(uint256[] memory _percentsTaxBuy, address[]
memory _addressesTaxBuy) public onlyOwner
function setTaxSell(uint256[] memory _percentsTaxSell,
address[] memory _addressesTaxSell) public onlyOwner
function setTaxTransfer(uint256[] memory _percentsTaxTransfer,
address[] memory _addressesTaxTransfer) public onlyOwner
```

Recommendation

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

RF - Redundant Functionality

Criticality	Minor / Informative
Location	SimpleToken.sol#L287
Status	Unresolved

Description

The contract inherits the `Ownable` functionality that enable Ownership privillages to the contract's owner. In the `Ownable` contract here is a function `owner()` that is public and returns the current owner. The `SimpleToken` contract has a public function `getOwner` that has exactly the same functionality, therefore it is redundant.

```
function getOwner() external view returns (address) {
    return owner();
}
```

Recommendation

It is recommended to remove redundant functionality to enhance code optimization and readability.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	SimpleToken.sol
Status	Unresolved

Description

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

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

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

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

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	SimpleToken.sol#L34
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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SimpleToken.sol#L154,186,190,194,198,202,206,210,220,230,273,274,275
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[] memory _percentsTax
address _addressLiquidity
address _addressIgnoreTax
address[] memory _addressesTaxBuy
uint256[] memory _percentsTaxBuy
address[] memory _addressesTaxSell
uint256[] memory _percentsTaxSell
uint256[] memory _percentsTaxTransfer
address[] memory _addressesTaxTransfer
uint8 public _decimals
string public _symbol
string public _name
```

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/stable/style-guide.html#naming-conventions.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	SimpleToken.sol#L360,372,384
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.

```
amountTax =
amount.div(100).mul(SwapBlock.getPercentsTaxSell()[i])
```

Recommendation

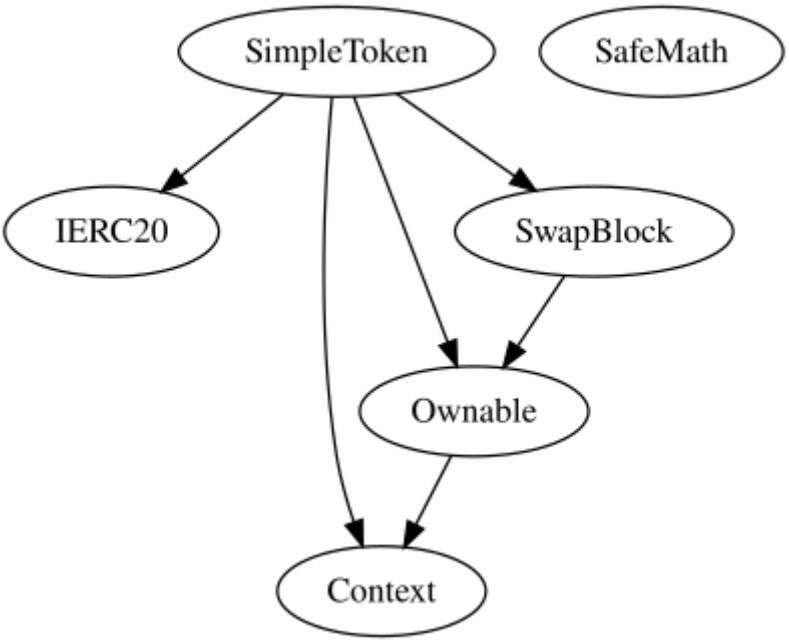
To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

Functions Analysis

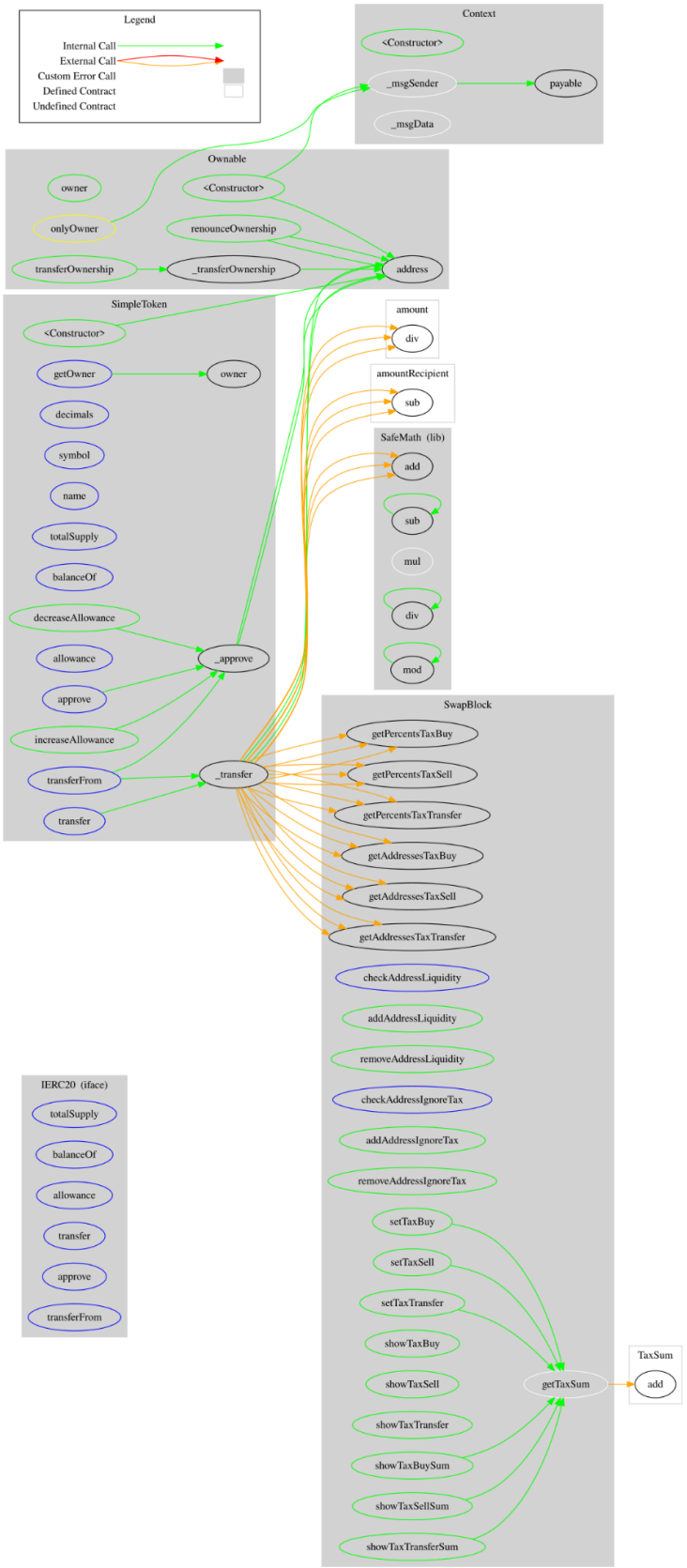
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SwapBlock	Implementation	Ownable		
	getTaxSum	Internal		
	getPercentsTaxBuy	Public		-
	getPercentsTaxSell	Public		-
	getPercentsTaxTransfer	Public		-
	getAddressesTaxBuy	Public		-
	getAddressesTaxSell	Public		-
	getAddressesTaxTransfer	Public		-
	checkAddressLiquidity	External		-
	addAddressLiquidity	Public	✓	onlyOwner
	removeAddressLiquidity	Public	✓	onlyOwner
	checkAddressIgnoreTax	External		-
	addAddressIgnoreTax	Public	✓	onlyOwner
	removeAddressIgnoreTax	Public	✓	onlyOwner
	setTaxBuy	Public	✓	onlyOwner
	setTaxSell	Public	✓	onlyOwner
	setTaxTransfer	Public	✓	onlyOwner
	showTaxBuy	Public		-
	showTaxSell	Public		-
	showTaxTransfer	Public		-

	showTaxBuySum	Public		-
	showTaxSellSum	Public		-
	showTaxTransferSum	Public		-
SimpleToken	Implementation	Context, Ownable, IERC20, SwapBlock		
		Public	✓	-
	getOwner	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_approve	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Context
<Constructor>
_msgSender
payable
_msgData
Ownable
owner
<Constructor>
onlyOwner
renounceOwnership
transferOwnership
_transferOwnership
address
SimpleToken
<Constructor>
getOwner
owner
decimals
symbol
name
totalSupply
balanceOf
decreaseAllowance
allowance
_approve
approve
increaseAllowance
transferFrom
_transfer
transfer
amount
div
amountRecipient
sub
SafeMath (lib)
add
sub
mul
div
mod
SwapBlock
getPercentsTaxBuy
getPercentsTaxSell
getPercentsTaxTransfer
getAddressesTaxBuy
getAddressesTaxSell
getAddressesTaxTransfer
checkAddressLiquidity
addAddressLiquidity
removeAddressLiquidity
checkAddressIgnoreTax
addAddressIgnoreTax
removeAddressIgnoreTax
setTaxBuy
setTaxSell
setTaxTransfer
showTaxBuy
showTaxSell
showTaxTransfer
showTaxBuySum
showTaxSellSum
showTaxTransferSum
getTaxSum
TaxSum
add
IERC20 (iface)
totalSupply
balanceOf
allowance
transfer
approve
transferFrom

Summary

INBUVE contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. INBUVE is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 1% fees.

Disclaimer

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

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

About Cyberscope

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

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

The Cyberscope team

cyberscope.io