Cyberscope

Audit Report

QuestLife

June 2024

Network	ARBITRUM
Address	0xe57141aa2e0c53def275c71602af9e394977e13e
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
●	IFDL	Incorrect Fee Deduction Logic	Unresolved
●	CR	Code Repetition	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	ITI	Inefficient Tax Implementation	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	SimpleToken
Compiler Version	v0.8.25+commit.b61c2a91
Optimization	200 runs
Explorer	https://arbiscan.io/address/0xe57141aa2e0c53def275c71602af9e394977e13e
Address	0xe57141aa2e0c53def275c71602af9e394977e13e
Network	ARBITRUM
Symbol	QLIFE
Decimals	18
Total Supply	10,000,000,000
Badge Eligibility	Yes

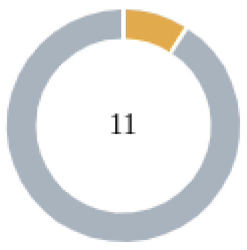
Audit Updates

Initial Audit	06 Jun 2024

Source Files

Filename	SHA256
SimpleToken.sol	14f8b41114d1111092696ac360a9ce7f7706c960beb567eb3e8379d67d6be4b8

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	10

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	10	0	0	0

IFDL - Incorrect Fee Deduction Logic

Criticality	Medium
Location	SimpleToken.sol#L360
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.

```
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#L344
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.

More specifically, the tax deduction, the `_balances` changes and the Transfer event emission in the `_transfer` has a lot of code repetition.

```
function _transfer(address sender, address recipient, uint256 amount)
internal {
    require(sender != address(0), "Transfer from the zero address");
    require(recipient != address(0), "Transfer to the zero address");
    require(amount <= _balances[sender], "Transfer amount exceeds
balance");

    _balances[sender] = _balances[sender].sub(amount);

    if (addressesIgnoreTax[sender] || addressesIgnoreTax[recipient]) {
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    } else {
        uint256 amountRecipient = amount;
        uint256 amountTax = 0;

        // checkAddressIgnoreTax

        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) {

            for (uint i; i < SwapBlock.getPercentsTaxSell().length; i++)
{
                amountTax =
amount.div(100).mul(SwapBlock.getPercentsTaxSell()[i]);
                amountRecipient = amountRecipient.sub(amountTax);
                _balances[SwapBlock.getAddressesTaxSell()[i]] =
SafeMath.add(_balances[SwapBlock.getAddressesTaxSell()[i]], amountTax);
                emit Transfer(sender,
SwapBlock.getAddressesTaxSell()[i], amountTax);
            }
            _balances[recipient] =
_balances[recipient].add(amountRecipient);
            emit Transfer(sender, recipient, amountRecipient);

        } else if (SwapBlock.getPercentsTaxTransfer().length > 0) {

            for (uint i; i < SwapBlock.getPercentsTaxTransfer().length;
i++) {
                amountTax =
amount.div(100).mul(SwapBlock.getPercentsTaxTransfer()[i]);
                amountRecipient = amountRecipient.sub(amountTax);
                _balances[SwapBlock.getAddressesTaxTransfer()[i]] =
SafeMath.add(_balances[SwapBlock.getAddressesTaxTransfer()[i]],
amountTax);
                emit Transfer(sender,
SwapBlock.getAddressesTaxTransfer()[i], amountTax);
            }

            _balances[recipient] =
_balances[recipient].add(amountRecipient);
            emit Transfer(sender, recipient, amountRecipient);

        } else {
            _balances[recipient] =
_balances[recipient].add(amountRecipient);
            emit Transfer(sender, recipient, amountRecipient);
        }
    }

}
```

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#L193,197,205,209,213,223,233
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#L283,284
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#L157,213,223,233
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 {
    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 {
    require(_percentsTaxSell.length == _addressesTaxSell.length,
"_percentsTaxSell.length != _addressesTaxSell.length");

    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 {
    require(_percentsTaxTransfer.length == _addressesTaxTransfer.length,
"_percentsTaxTransfer.length != _addressesTaxTransfer.length");

    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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	SimpleToken.sol#L193,197,205,209,213,223,233
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.

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/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	SimpleToken.sol#L193,197,205,209
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function addAddressLiquidity(address _addressLiquidity) public onlyOwner
function removeAddressLiquidity (address _addressLiquidity) public
onlyOwner
function addAddressIgnoreTax(address _addressIgnoreTax) public onlyOwner
function removeAddressIgnoreTax (address _addressIgnoreTax) public
onlyOwner
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SimpleToken.sol#L157,189,193,197,201,205,209,213,223,233,276,277,278
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/v0.8.17/style-guide.html#naming-convention.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	SimpleToken.sol#L363,375,387
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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	SimpleToken.sol#L13
Status	Unresolved

Description

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

```
pragma solidity >=0.8.19;
```

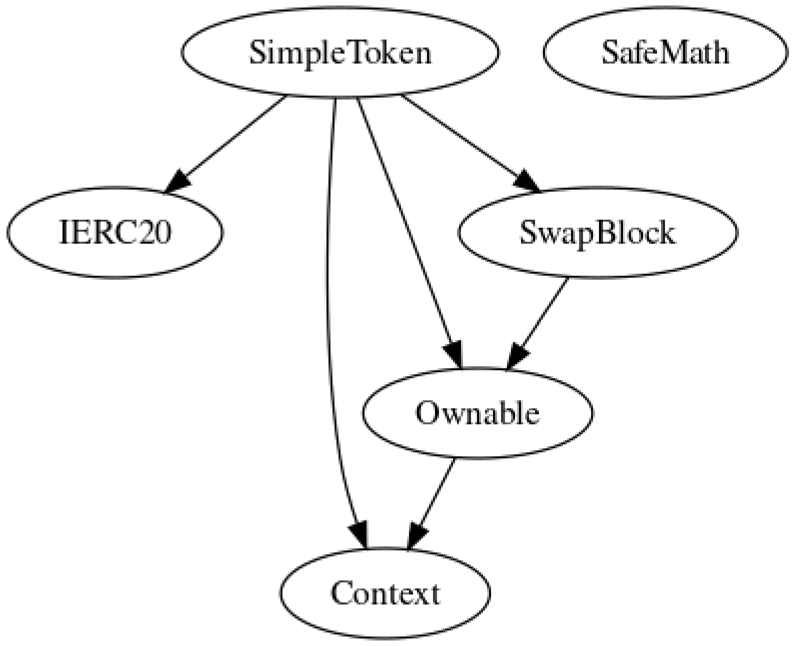
Recommendation

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

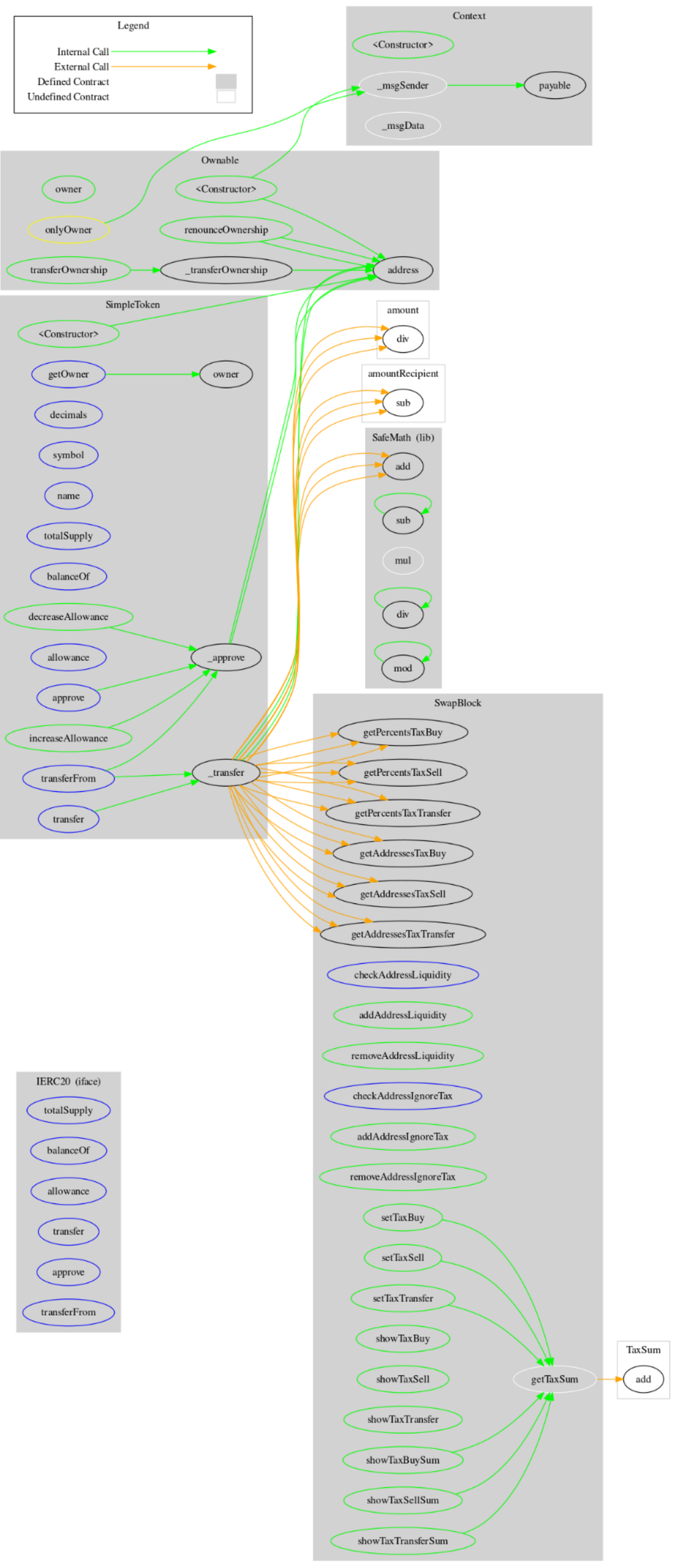
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
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
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
getTaxSum
showTaxTransfer
showTaxBuySum
showTaxSellSum
showTaxTransferSum
TaxSum
add
IERC20 (iface)
totalSupply
balanceOf
allowance
transfer
approve
transferFrom

Summary

QuestLife is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. There is also a limit of max 1% fees. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

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