Cyberscope

Audit Report

ASTRO TOKEN

September 2023

Network	BSC
Address	0x8c2a02f49d825645d447aca91483c865ef79fc91
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
●	RRS	Redundant Require Statement	Unresolved
●	RCC	Redundant Condition Check	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RVD	Redundant Variable Declaration	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
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	AstroToken
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0x8c2a02f49d825645d447aca91483c865ef79fc91
Address	0x8c2a02f49d825645d447aca91483c865ef79fc91
Network	BSC
Symbol	ASTRO
Decimals	18
Total Supply	10,000,000

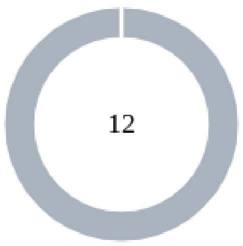
Audit Updates

Initial Audit	10 Sep 2023

Source Files

Filename	SHA256
AstroToken.sol	57181016fd8fa500c6946388364a71d7e9a26272fe60d663def807ba66b1ca06

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	12	0	0	0

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	AstroToken.sol#L717
Status	Unresolved

Description

The smart contract incorporates a redundant require statement within the `_mint` function that verifies whether the `mintingFinishedPermanent` variable is set to `false`. However, this verification is redundant since the `_mint` function is only invoked within the constructor of the contract. Given that constructors are executed only once during contract deployment and the `_mint` function is declared as `internal virtual`, the `_mint` function will never be called again. Therefore, the require statement serves no functional purpose and only adds unnecessary complexity to the contract.

```
    bool public mintingFinishedPermanent = false;

     constructor (address creator_,string memory name_, string memory
symbol_,uint8 decimals_, uint256 tokenSupply_, uint8 liquidityFee_,
address liquidity_, uint8 marketingFee_, address marketing_) {
        ...
        _mint(_creator,tokenSupply_);
        mintingFinishedPermanent = true;
    }

    function _mint(address account, uint256 amount) internal virtual {
        require(!mintingFinishedPermanent,"cant be minted anymore!");
        ...
    }
```

Recommendation

It is recommended to remove the redundant `require` statement that verifies the `mintingFinishedPermanent` variable. Additionally, the `mintingFinishedPermanent` variable itself can be eliminated from the contract, as it serves no functional purpose. By doing so, the contract will be simplified, making it easier to understand and maintain.

RCC - Redundant Condition Check

Criticality	Minor / Informative
Location	AstroToken.sol#L689
Status	Unresolved

Description

The contract checks whether the sender or the recipient is excluded from fees using an initial if statement. If neither the sender nor the recipient is excluded from fees, the code within the second if statement is executed. However, the condition in the second if statement redundantly checks for the opposite of the first if statement. This use of two separate `if` statements, instead of a more straightforward `if-else` structure, introduces unnecessary complexity and increases the potential for errors. It also increases the gas cost associated with executing this portion of the smart contract, which could be a concern in a high-throughput environment.

```
   if(_isExcludedFromFee[sender] || _isExcludedFromFee[recipient]){
        _balances[sender] = senderBalance - amount;
        _balances[recipient] += amount;
        emit Transfer(sender, recipient, amount);
    }
    if(!_isExcludedFromFee[sender] && !_isExcludedFromFee[recipient]){
        uint256 amountForLiquidity = (amount * _liquidityFee) / 100;
        uint256 amountForMarketing = (amount * _marketingFee) / 100;

        ...
     }
```

Recommendation

The team is advised to refactor the code to use an `if-else` structure instead of two separate `if` statements. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	AstroToken.sol#L638,643,648,653
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.

The contract does not emit events on the functions `excludeFromFee` , `includeInFee` , `setMarketingFeePercent` , and `setLiquidityFeePercent` .

```
 function excludeFromFee(address account) public onlyOwner {
        require(!_isExcludedFromFee[account], "Account is already 
excluded");
        _isExcludedFromFee[account] = true;
    }

    function includeInFee(address account) public onlyOwner {
        require(_isExcludedFromFee[account], "Account is already 
included");
        _isExcludedFromFee[account] = false;
    }

    function setMarketingFeePercent(uint8 marketingFee) external 
onlyOwner() {
        require(marketingFee >= 0 && marketingFee <= 100,"out of 
range");
        _marketingFee = marketingFee;
    }

    function setLiquidityFeePercent(uint8 liquidityFee) external 
onlyOwner() {
        require(liquidityFee >= 0 && liquidityFee <= 100,"out of 
range");
        _liquidityFee = liquidityFee;
    }
```

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.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	AstroToken.sol#L454
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 `mintedByAstroVerse` variable to `true`, but the variable is not used in a meaningful way by the contract. As a result, this variable is redundant.

```
bool public mintedByAstroVerse = 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	AstroToken.sol#L477,478,479,480,482,484,491
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.

```
_name
_symbol
_decimals
_creator
_marketing
_liquidity
mintingFinishedPermanent
```

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	AstroToken.sol#L454
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.

```
bool public mintedByAstroVerse = true
```

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	AstroToken.sol#L462,463,465,466,469,658
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.

```
uint8 public _liquidityFee
uint8 public _marketingFee
address public _marketing
address public _liquidity
address public _creator
uint256 _value
```

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	AstroToken.sol#L474
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.

```
mapping (address => bool) private _isdevWallet
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	AstroToken.sol#L649,654
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(marketingFee >= 0 && marketingFee <= 100,"out of range")
require(liquidityFee >= 0 && liquidityFee <= 100,"out of range")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	AstroToken.sol#L244,268,293,303,322,332,349,359,374,384,399,423,435
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 isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize/address.code.length, which
returns 0
        // for contracts in construction, since the code is only stored
at the end
        // of the constructor execution.

        return account.code.length > 0;
    }

function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	AstroToken.sol#L480,482,484
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.

```
_creator = creator_
_marketing = marketing_
_liquidity = liquidity_
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	AstroToken.sol#L440
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Functions Analysis

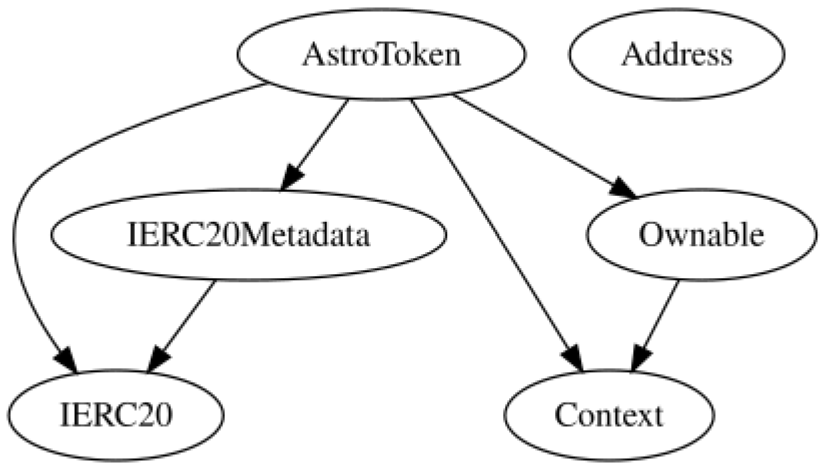
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		

		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
AstroToken	Implementation	Context, IERC20, IERC20Meta		

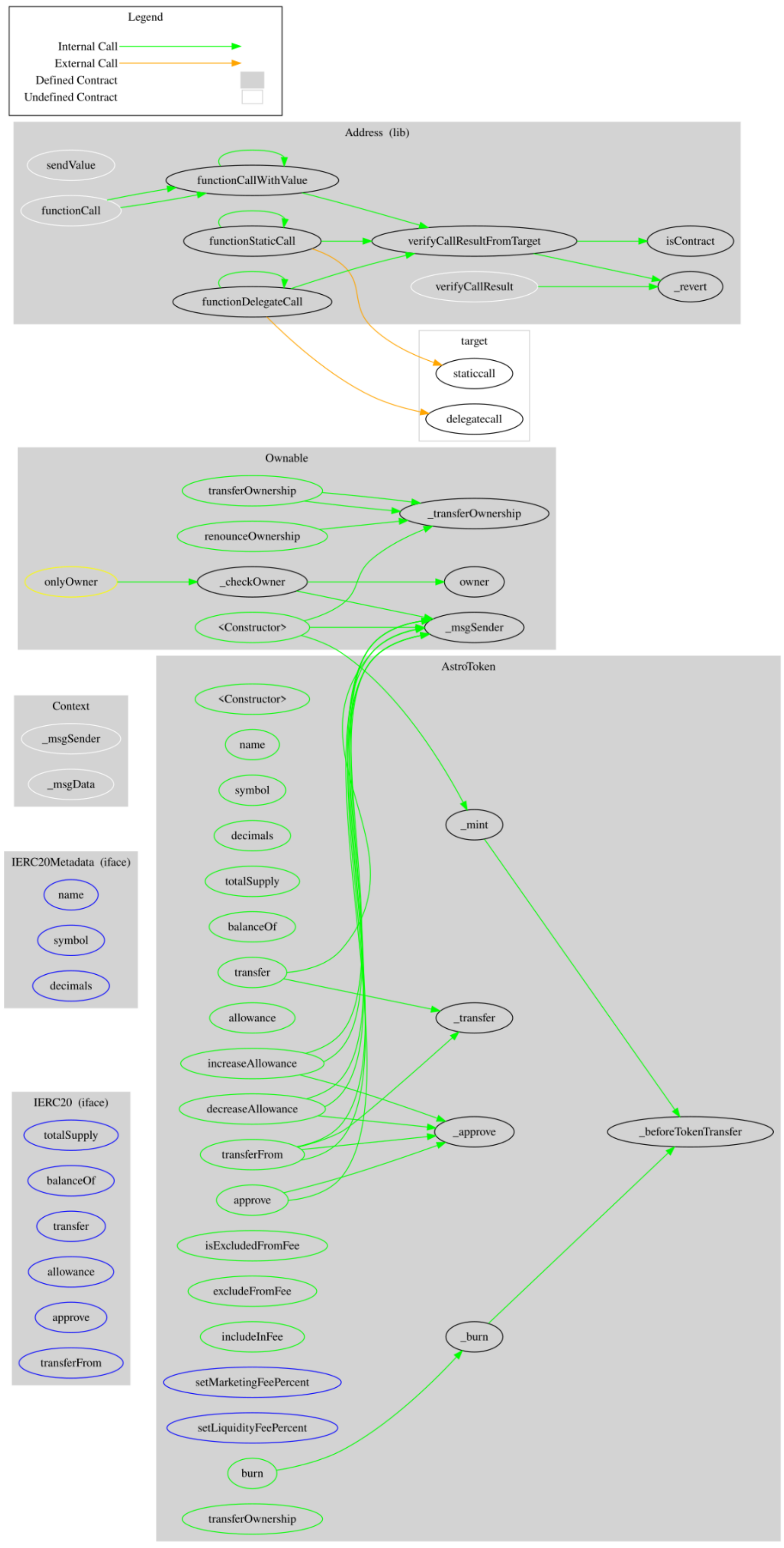
		data, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromFee	Public		-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setMarketingFeePercent	External	✓	onlyOwner
	setLiquidityFeePercent	External	✓	onlyOwner
	burn	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	

	_beforeTokenTransfer	Internal	✓	
	transferOwnership	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
verifyCallResult
_revert
functionDelegateCall
target
staticcall
delegatecall
Ownable
transferOwnership
_transferOwnership
renounceOwnership
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
AstroToken
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
increaseAllowance
decreaseAllowance
transferFrom
approve
isExcludedFromFee
excludeFromFee
includeInFee
setMarketingFeePercent
setLiquidityFeePercent
burn
transferOwnership
_mint
_transfer
_approve
_beforeTokenTransfer
_burn
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

ASTRO TOKEN contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. ASTRO TOKEN is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

https://bscscan.com/tx/0x8b99ff5259e274a0a4ae86d30d8a252df28db0eab281bb09fdaad478d75034db

The fees are locked at 3% for all transfers.

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