Cyberscope

Audit Report

ACG

June 2023

Network	BSC
Address	0xAa7B539A052a78a15eC145a3c29Aec293dbC1eE0
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
●	IAE	Invalid Arithmetic Expression	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	ACG
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0xaa7b539a052a78a15ec145a3c29aec293dbc1ee0
Address	0xaa7b539a052a78a15ec145a3c29aec293dbc1ee0
Network	BSC
Symbol	ACG
Decimals	18
Total Supply	1,000,000,000

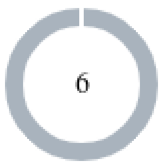
Audit Updates

Initial Audit	23 May 2023 https://github.com/cyberscope-io/audits/blob/main/acg/v1/audit.pdf
Corrected Phase 2	28 May 2023 https://github.com/cyberscope-io/audits/blob/main/acg/v2/audit.pdf
Corrected Phase 3	05 Jun 2023

Source Files

Filename	SHA256
ACG.sol	143634e05ad6e9c56ae528b31cb9241a7d354d328632ee145c494e58a604cec5

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	6

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	6	0	0	0

IAE - Invalid Arithmetic Expression

Criticality	Minor / Informative
Location	ACG.sol#L1740,1742,1746,1749,1761,1764
Status	Unresolved

Description

From our understanding, the `10e18` expression represents the token decimals. The decimal points of the token are 18. The contract normalizes certain values to the contract's decimals. However, this is done incorrectly since the `1018` does not equal 18 decimal places, but 19. As a result, the arithmetic operations that involve an arithmetic operation with the `10e18` will produce inaccurate results.

```
uint256 _vLimiter = 5000000 * 10e18
newLimit * 10e18 <= _vLimiter
newLimit * 10e18 != swapThresholdLimit
...
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the contract. The team is advised to modify these segments to correctly represent the decimal points of the contract. A recommended approach would be to modify the `10e18` expression to `1e18`.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ACG.sol#L9,168,169,186,532
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	ACG.sol#L286,311,340,371,381,396,406,445,639,765,776,791,800,813,826,865,1131,1164,1190,1221,1269,1289
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 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");
    }

function functionCall(address target, bytes memory data) internal returns
(bytes memory) {
        return functionCallWithValue(target, data, 0, "Address: low-level
call failed");
    }

...
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	ACG.sol#L462
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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	ACG.sol#L5,104,151,207,230,477,539,566,651,736,880,910,1299
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity >=0.6.2;
pragma solidity >=0.5.0;
pragma solidity ^0.8.1;
pragma solidity ^0.8.0;
pragma solidity =0.8.17;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	ACG.sol#L230,477,539,566,651,736,880,910
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 ^0.8.1;
pragma solidity ^0.8.0;
```

Recommendation

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

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-

	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-

	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-

	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
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		
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-

Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	

	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	forceApprove	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
	_callOptionalReturnBool	Private	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-

	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ACG	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
		External	Payable	-
	_approve	Internal	✓	
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-

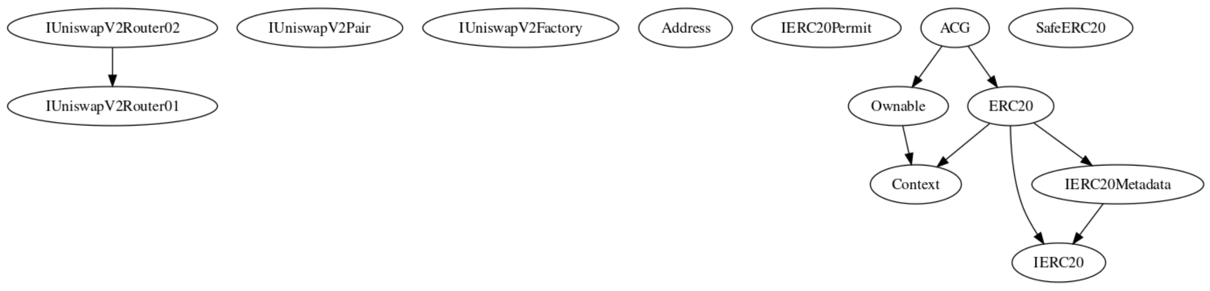
	getBNBBalance	Public		-
	getErc20TokenFeeBalance	Public		-
	getErc20TokensBalance	Public		-
	setSwapRouter	External	✓	onlyOwner
	setSwapThresholdLimit	External	✓	onlyOwner
	setPenaltyTxAmount	External	✓	onlyOwner
	setErc20TokenAddress	External	✓	onlyOwner
	setAntiBotStatus	External	✓	onlyOwner
	setMaxBotSellCount	External	✓	onlyOwner
	setBotSellTimeLimit	External	✓	onlyOwner
	setOperationWallet	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setPoolsLeaderboardWallet	External	✓	onlyOwner
	setCommunityWallet	External	✓	onlyOwner
	setTreasuryOneWallet	External	✓	onlyOwner
	setTreasuryTwoWallet	External	✓	onlyOwner
	setPenaltyWallet	External	✓	onlyOwner
	_updateFees	Private	✓	
	_calculateTotalNewFees	Internal		
	setOperationFeePercent	External	✓	onlyOwner feesNotBeingSet
	setMarketingFeePercent	External	✓	onlyOwner feesNotBeingSet
	setLiquidityFeePercent	External	✓	onlyOwner feesNotBeingSet

	setPoolsLeaderboardFeePercent	External	✓	onlyOwner feesNotBeingSet
	setCommunityFeePercent	External	✓	onlyOwner feesNotBeingSet
	setTreasuryOneFeePercent	External	✓	onlyOwner feesNotBeingSet
	setTreasuryTwoFeePercent	External	✓	onlyOwner feesNotBeingSet
	setPenaltyFeePercent	External	✓	onlyOwner feesNotBeingSet
	_removeAllFees	Private	✓	
	_restoreAllFees	Private	✓	
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	_calculateFees	Private		
	_calculateBuyFee	Private		
	_calculateSellFee	Private		
	_calculateFee	Private		
	_getCurrentSupply	Private		
	_getRate	Private		
	tokenFromReflection	Public		-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		

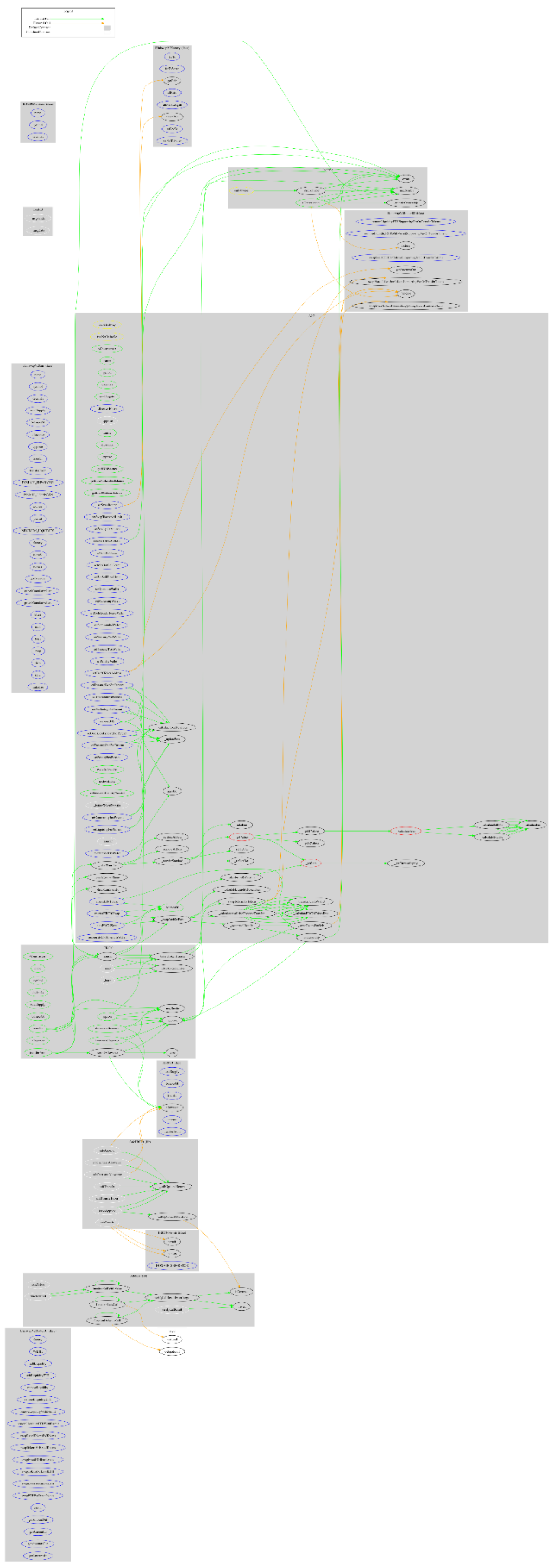
	_getRValues	Private		
	_calculateERC20TokenFees	Private		
	_calculateLiquidityTokenFees	Private		
	_takeFees	Private	✓	
	_beforeTokenTransfer	Internal		
	_checkCanTransfer	Internal		
	_checkAntibotStatus	Internal		
	_transfer	Internal	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_reflectBot	Private	✓	
	_swapAndGetFees	Private	✓	lockTheSwap
	_swapAndLiquify	Private	✓	
	_calculateAvailableFeesAndTransfer	Private	✓	
	_transferFeesToWallet	Private	✓	
	_swapTokensForBnb	Private	✓	
	_addLiquidity	Private	✓	
	_swapTokensForTokens	Private	✓	
	manualBNBSwap	External	✓	onlyOwner
	manualERC20Swap	External	✓	onlyOwner lockTheSwap
	autoERC20Swap	External	✓	onlyOwner
	recoverBNB	External	✓	onlyOwner
	recoverBNBToWallet	External	✓	onlyOwner

	recoverERC20Tokens	External	✓	onlyOwner
	recoverERC20TokensToWallet	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

ACG contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. ACG 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 15% buy fees and 25% if the transferred amount is a specific threshold that is defined by the contract owner. Lastly, the contract has an antibot throttling mechanism that can prevent the transfers of up to 100 blocks.

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