Cyberscope

Audit Report

BabyChita

February 2023

Type	BEP20
Network	BSC
Address	0x6859b546FB887fb5018AE0cd01DA0fff2B3f5Bc7
Audited by	© cyberscope

Table of Contents

Review

Contract Name	CHITAVERSE
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6859b546fb887fb5018ae0cd01da0fff2b3f5bc7
Address	0x6859b546fb887fb5018ae0cd01da0fff2b3f5bc7
Network	BSC
Symbol	BCT
Decimals	9
Total Supply	10,000,000,000

Audit Updates

Initial Audit	04 Feb 2023

Source Files

Filename	SHA256
CHITAVERSE.sol	cd75e59627ba8484776e2b83c8d9d61883c829441bf42bf8182098b745fa456e

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

ST - Stops Transactions

Criticality	Critical
Location	CHITAVERSE.sol#L283,292
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `cooldownTimerInterval` to a high value. As a result, the contract may operate as a honeypot.

```
require(cooldownTimer[recipient] < block.timestamp,"Please wait for 1min
between two operations");
cooldownTimer[recipient] = block.timestamp + cooldownTimerInterval;
```

The contract owner can stop the transfers for all users excluding the some of them. The owner may take advantage of it by setting the `_maxTxAmount` to zero.

```
require(amount <= _maxTxAmount || isTxLimitExempt[sender], "TX Limit
Exceeded");
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. The contract should also prevent the `cooldownTimerInterval` to be more than a reasonable amount. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	CHITAVERSE.sol#L414,432
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setFees` or `setSellMultiplier` function with a high percentage value.

```
function setFees(uint256 _liquidityFee, uint256 _marketingFee, uint256
_devFee, uint256 _feeDenominator) external onlyOwner {
    liquidityFee = _liquidityFee;
    marketingFee = _marketingFee;
    devFee = _devFee;
    totalFee = _liquidityFee.add(_marketingFee).add(_devFee);
    feeDenominator = _feeDenominator;
}

function setSellMultiplier(uint256 multiplier) external onlyOwner{
    _sellMultiplier = multiplier;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

BC - Blacklists Addresses

Criticality	Medium
Location	CHITAVERSE.sol#L285
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `isBots` function.

```
require(!isBot[recipient] && !isBot[sender], 'Address is excluded.');
```

Recommendation

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	ZD	Zero Division	Unresolved
Minor / Informative	PVC	Price Volatility Concern	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

ZD - Zero Division

Criticality	Critical
Location	CHITAVERSE.sol#L347
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

The variable `totalFee` can be set to zero and produce a zero division revert.

```
uint256 amountToLiquify =
contractTokenBalance.mul(liquidityFee).div(totalFee).div(2);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero or should not allow executing of the corresponding statements.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	CHITAVERSE.sol#L338
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	CHITAVERSE.sol#L181,182,183,184,190
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.

```
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
address routerAddress = 0x10ED43C718714eb63d5aA57B78B54704E256024E
uint256 _totalSupply = 10000000000 * (10 ** _decimals)
```

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	CHITAVERSE.sol#L83,133,181,182,183,186,187,188,190,191,192,194,195,207,414,422,435,444,449,457
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.

```
mapping (address => bool) internal _intAddr
function WETH() external pure returns (address);
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
string constant _name = "BabyChita Token"
string constant _symbol = "BCT"
uint8 constant _decimals = 9
uint256 _totalSupply = 10000000000 * (10 ** _decimals)
uint256 public _maxTxAmount = (_totalSupply * 100 ) / 100
uint256 public _maxWalletSize = (_totalSupply * 100) / 100
mapping (address => uint256) _balances
mapping (address => mapping (address => uint256)) _allowances
uint256 public _sellMultiplier = 1

...
```

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	CHITAVERSE.sol#L415,433,441,446
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.

```
liquidityFee = _liquidityFee
_sellMultiplier = multiplier
_maxTxAmount = amountBuy
swapThreshold = _amount
```

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	CHITAVERSE.sol#L388
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 buyTokens(uint256 amount, address to) internal swapping {
        address[] memory path = new address[](2);
        path[0] = WBNB;
        path[1] = address(this);

        router.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
amount}(
            0,
            path,
            to,
            block.timestamp
        );
    }
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CHITAVERSE.sol#L218
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 public swapThreshold = _totalSupply / 10000 * 50
```

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	CHITAVERSE.sol#L109,436,437
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.

```
owner = adr
marketingFeeReceiver = _marketingFeeReceiver
devFeeReceiver = _devFeeReceiver
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	CHITAVERSE.sol#L18
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 ^0.8.7;
```

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.

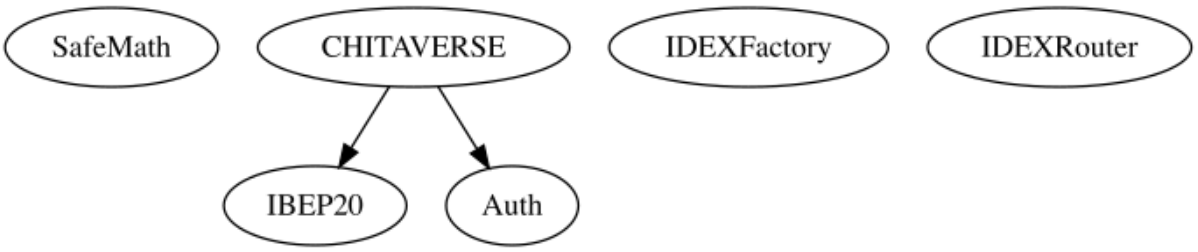
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	isOwner	Public		-
	transferOwnership	Public	✓	onlyOwner

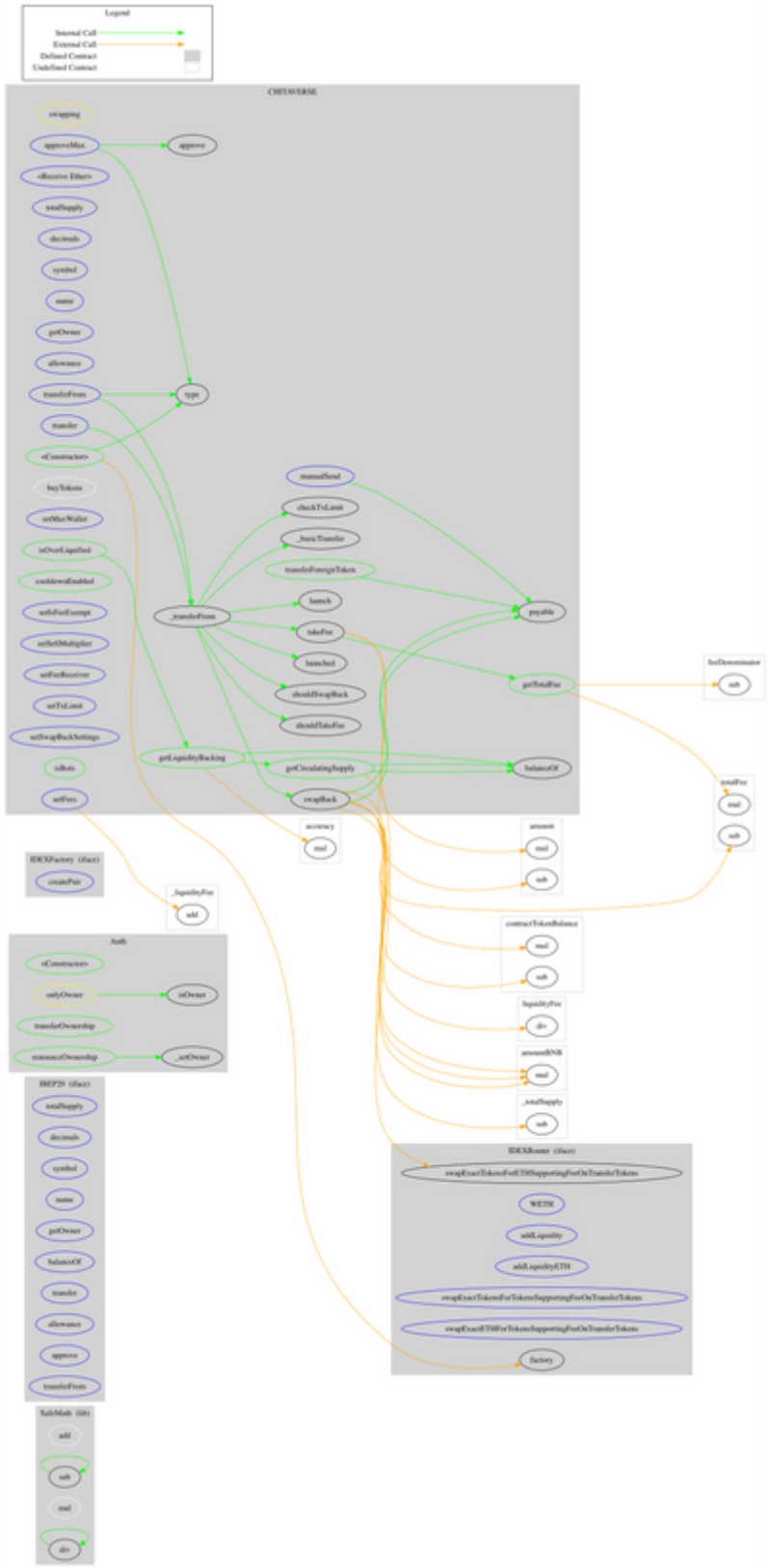
	_setOwner	Private	✓	
	renounceOwnership	Public	✓	onlyOwner
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
CHITAVERSE	Implementation	IBEP20, Auth		
		Public	✓	Auth
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-

	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	checkTxLimit	Internal		
	shouldTakeFee	Internal		
	getTotalFee	Public		-
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	buyTokens	Internal	✓	swapping
	launched	Internal		
	launch	Internal	✓	
	setMaxWallet	External	✓	onlyOwner
	setFees	External	✓	onlyOwner
	cooldownEnabled	Public	✓	onlyOwner
	setIsFeeExempt	External	✓	onlyOwner
	setSellMultiplier	External	✓	onlyOwner
	setFeeReceiver	External	✓	onlyOwner
	setTxLimit	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	isBots	Public	✓	onlyOwner
	manualSend	External	✓	-
	transferForeignToken	Public	✓	-
	getCirculatingSupply	Public		-
	getLiquidityBacking	Public		-
	isOverLiquified	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
CHITAVERSE
swapping
approveMax
approve
«Receive Ether»
totalSupply
decimals
symbol
name
getOwner
allowance
transferFrom
type
transfer
«Constructor»
buyTokens
setMaxWallet
isOverLiquified
cooldownEnabled
setIsFeeExempt
setSellMultiplier
setFeeReceiver
setTxLimit
setSwapBackSettings
isBots
setFees
manualSend
checkTxLimit
_basicTransfer
transferForeignToken
_transferFrom
launch
takeFee
launched
shouldSwapBack
shouldTakeFee
payable
getTotalFee
feeDenominator
sub
getLiquidityBacking
getCirculatingSupply
balanceOf
swapBack
totalFee
mul
sub
accuracy
mul
amount
mul
sub
IDEXFactory (iface)
createPair
_liquidityFee
add
Auth
«Constructor»
onlyOwner
isOwner
transferOwnership
renounceOwnership
_setOwner
contractTokenBalance
mul
sub
liquidityFee
div
amountBNB
mul
_totalSupply
sub
IBEP20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
SafeMath (lib)
add
sub
mul
div
IDEXRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidity
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
factory

Summary

There are some functions that can be abused by the owner like stop transactions, manipulate the fees and blacklist addresses. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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