Cyberscope

Audit Report

Kita Inu

May 2023

Network BSC

Address 0xD8CC2D4fE120506dF41BC83d5a061875660d6f63

Audited by © cyberscope

Table of Contents

Review

Contract Name	KITA
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0xd8cc2d4fe120506df41bc83d5a061875660d6f63
Address	0xd8cc2d4fe120506df41bc83d5a061875660d6f63
Network	BSC
Symbol	KITA
Decimals	18
Total Supply	1.000.000.000

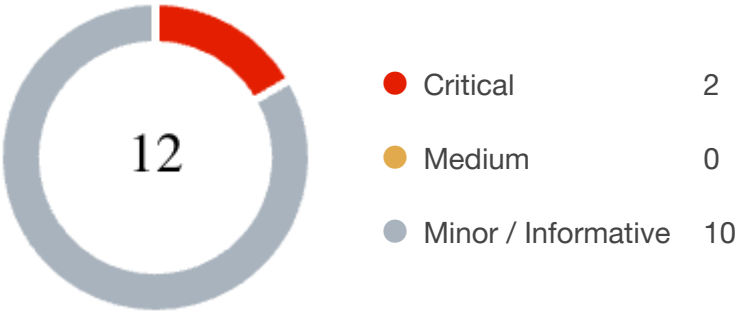
Audit Updates

Initial Audit	25 May 2023

Source Files

Filename	SHA256
KITA.sol	94bb941763cfe40df1baf355e96b5bcfdfb5d4ccc6453fe871fea6922a09c6cf

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	TFI	Transfer Fees Inconsistency	Unresolved
●	MLF	Misleading Liquidity Functionality	Unresolved
●	MU	Modifiers Usage	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MSE	Missing Solidity Events	Unresolved
●	RCV	Redundant Constant Variable	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
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

MLF - Misleading Liquidity Functionality

Criticality	Critical
Location	KITA.sol#L29,134,155
Status	Unresolved

Description

The contract implements a liquidity functionality, indicating that it intends to interact with liquidity pools. However, there is no implementation or function present that enables users or the contract itself to add liquidity to the desired pool. Additionally, the tokens that are sent to the `pancakeSwapRouter` are locked there and the `pancakeSwapRouter` address does not implement any method to be able to execute the `updatePancakeSwapRouter()` .

```
address private pancakeSwapRouter =
address(0x10ED43C718714eb63d5aA57B78B54704E256024E);

uint256 liquidityTax = taxAmount - burnTax - charityTax - marketingTax;
if (liquidityTax > 0) {
    _transferToAddress(sender, pancakeSwapRouter, liquidityTax);
}

function updatePancakeSwapRouter(address newRouter) external {
    require(msg.sender == pancakeSwapRouter, "Caller is not the current
PancakeSwap router");
    pancakeSwapRouter = newRouter;
}
```

Recommendation

The contract should revisit the liquidity functionality since sending tokens to the router address does not have any effect.

TFI - Transfer Fees Inconsistency

Criticality	Critical
Location	KITA.sol#L94,109
Status	Unresolved

Description

The contract incorporates a fee mechanism to deduct a certain percentage from the sender's transaction amount. However, after deducting the fee, the contract proceeds to transfer the tax distributions from the sender to designated addresses by utilizing the `_transferToAddress` method. This double deduction of amounts results in an inconsistency between the actual transferred amount and the amount the sender expects to send.

```
function _transfer(address sender, address recipient, uint256 amount) private
{
    ...
    if (taxAmount > 0) {
        _handleTax(sender, taxAmount);
    }
    ...
}

function _handleTax(address sender, uint256 taxAmount) private {
    uint256 burnTax = (taxAmount * burnTaxPercentage) / taxPercentage;
    uint256 charityTax = (taxAmount * charityTaxPercentage) / taxPercentage;
    uint256 marketingTax = (taxAmount * marketingTaxPercentage) /
taxPercentage;

    if (burnTax > 0) {
        _transferToAddress(sender, burnAddress, burnTax);
    }
    if (charityTax > 0) {
        _transferToAddress(sender, charityWallet, charityTax);
    }
    if (marketingTax > 0) {
        _transferToAddress(sender, marketingWallet, marketingTax);
    }

    uint256 liquidityTax = taxAmount - burnTax - charityTax - marketingTax;
    if (liquidityTax > 0) {
        _transferToAddress(sender, pancakeSwapRouter, liquidityTax);
    }
}
```

Recommendation

It is recommended to review and revise the fee and tax distribution mechanisms to avoid double deductions from the sender's account. Consider implementing a streamlined approach where the fee and tax distribution are deducted only once, ensuring the transferred amount remains consistent with the sender's intent.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	KITA.sol#L160,165,170
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(msg.sender == deployerWallet || msg.sender ==
developerWallet, "Caller is not authorized");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	KITA.sol#L154,159,164,169
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 updates variables even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function updatePancakeSwapRouter(address newRouter) external {
      require(msg.sender == pancakeSwapRouter, "Caller is not
the current PancakeSwap router");
      pancakeSwapRouter = newRouter;
}

function updateExemptWallet(address wallet, bool isExempt)
external {
      require(msg.sender == deployerWallet || msg.sender ==
developerWallet, "Caller is not authorized");
      exemptWallets[wallet] = isExempt;
}

function enableTax() external {
      require(msg.sender == deployerWallet || msg.sender ==
developerWallet, "Caller is not authorized");
      isTaxEnabled = true;
}

function disableTax() external {
      require(msg.sender == deployerWallet || msg.sender ==
developerWallet, "Caller is not authorized");
      isTaxEnabled = false;
}
```

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.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	KITA.sol#L154,159,164,169
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 updatePancakeSwapRouter(address newRouter) external {
    require(msg.sender == pancakeSwapRouter, "Caller is not the
current PancakeSwap router");
    pancakeSwapRouter = newRouter;
}

function updateExemptWallet(address wallet, bool isExempt)
external {
    require(msg.sender == deployerWallet || msg.sender ==
developerWallet, "Caller is not authorized");
    exemptWallets[wallet] = isExempt;
}

function enableTax() external {
    require(msg.sender == deployerWallet || msg.sender ==
developerWallet, "Caller is not authorized");
    isTaxEnabled = true;
}

function disableTax() external {
    require(msg.sender == deployerWallet || msg.sender ==
developerWallet, "Caller is not authorized");
    isTaxEnabled = false;
}
```

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.

RCV - Redundant Constant Variable

Criticality	Minor / Informative
Location	KITA.sol#L37
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 `liquidityTaxPercentage` variable is not used in the contracts implementation. Hence, it is redundant.

```
uint256 private constant liquidityTaxPercentage = 5;
```

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. It is recommended to remove redundant variables.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	KITA.sol#L41,42,43,44
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
name
symbol
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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	KITA.sol#L26,27,28,29,30
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 private burnAddress =
address(0x000000000000000000000000000000000000dEaD)
address private charityWallet =
address(0xA67dE2b8c36848802b711D551c23935d987ABBEd)
address private marketingWallet =
address(0x9488cA8E59D7D68a63babB98Cb722AA7fcda3dfc)
address private deployerWallet =
address(0x31DaFbfb3f96f9E85518B6F2Afa508B76CE50386)
address private developerWallet =
address(0xefACd388769531AEa7546aF7A411fEA40cA434B2)
```

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	KITA.sol#L32,33,34,35,36
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 private constant taxPercentage = 8
uint256 private constant liquidityTaxPercentage = 5
uint256 private constant burnTaxPercentage = 1
uint256 private constant charityTaxPercentage = 1
uint256 private constant marketingTaxPercentage = 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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	KITA.sol#L33
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.

```
uint256 private constant liquidityTaxPercentage = 5
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	KITA.sol#L152
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.

```
pancakeSwapRouter = newRouter
```

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	KITA.sol#L2
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 ^0.8.0;
```

Recommendation

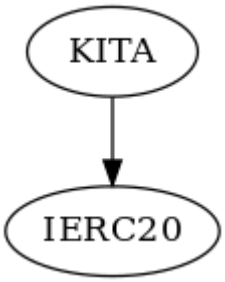
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

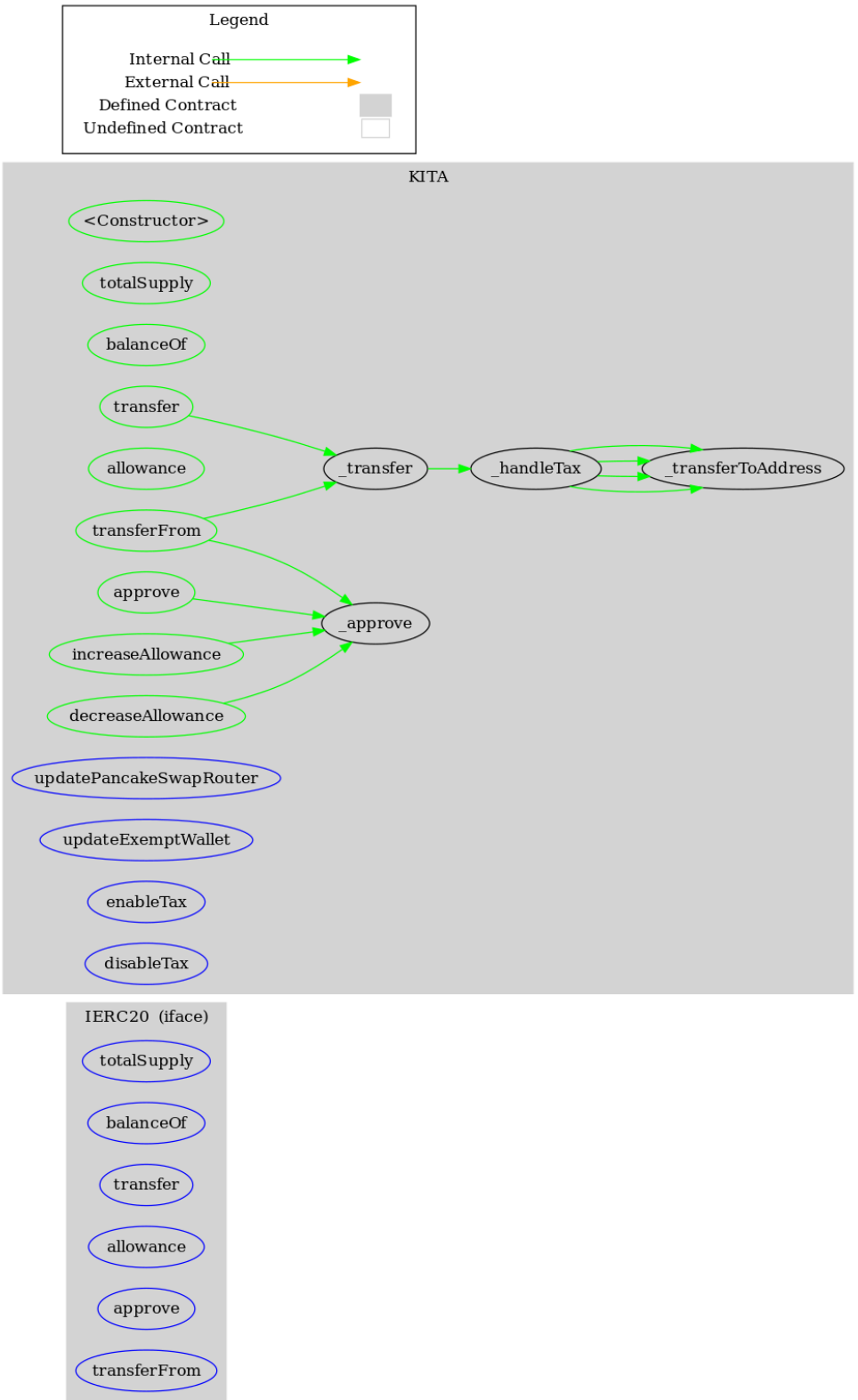
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
KITA	Implementation	IERC20		
		Public	✓	-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-

	_transfer	Private	✓	
	_handleTax	Private	✓	
	_transferToAddress	Private	✓	
	_approve	Private	✓	
	updatePancakeSwapRouter	External	✓	-
	updateExemptWallet	External	✓	-
	enableTax	External	✓	-
	disableTax	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
KITA
<Constructor>
totalSupply
balanceOf
transfer
allowance
transferFrom
approve
increaseAllowance
decreaseAllowance
updatePancakeSwapRouter
updateExemptWallet
enableTax
disableTax
_transfer
_handleTax
_transferToAddress
_approve
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Kita Inu contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Kita Inu is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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 fixed fee of 8% fees. Additionally, the taxes can be disabled or enabled.

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