Audit Report

ShibaKeanu

February 2024

SHA256 5f77ab2c3a3f2983ce9c881ce9cfa091b1b7d38e6573f62363ca40640bb2abf3

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
Pass	ST	Stops Transactions	Passed
Pass	OTUT	Transfers User's Tokens	Passed
Pass	ELFM	Exceeds Fees Limit	Passed
Pass	MT	Mints Tokens	Passed
Pass	BT	Burns Tokens	Passed
Pass	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	IBD	Inefficient Balance Deduction	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	ShibaKeanu
Testing Deploy	https://testnet.bscscan.com/address/0x215009632fa65d8046f275be0a2e8f7803066050
Symbol	SHIBK
Decimals	18
Total Supply	888,000,000,000,000
Badge Eligibility	Must Fix Criticals

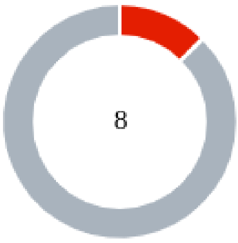
Audit Updates

Initial Audit	14 Feb 2024

Source Files

Filename	SHA256
contracts/ShibaKeanu.sol	5f77ab2c3a3f2983ce9c881ce9cfa091b1b7d38e6573f62363ca40640bb2abf3

Findings Breakdown



- Critical 1
- Medium 0
- Minor / Informative 7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

TSD - Total Supply Diversion

Criticality	Critical
Location	contracts/ShibaKeanu.sol#L550
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

Specifically, the contract is incorrectly setting the `_balances[from]` twice within separate unchecked blocks, leading to an unintended consequence in the balance deduction logic. Initially, it deducts the `_receiverAmount` from the sender's (from) balance. Subsequently, it incorrectly resets the sender's balance by deducting only the `_taxAmount` from the original balance (fromBalance), as shown in the second unchecked block. This sequential deduction and reset approach results in the sender's balance being effectively reduced by only the `_taxAmount`, disregarding the previously deducted `_receiverAmount`. Consequently, this oversight causes the contract to fail in accurately accounting for the total amount that should be deducted from the sender, comprising both the amount sent to the receiver and the tax amount, potentially leading to a significant discrepancy in the intended token distribution and balance management.

```
unchecked {
    _balances[from] = fromBalance - _recieverAmount;
    }
    _balances[to] += _recieverAmount;
        emit Transfer(from, to, _recieverAmount);
    unchecked {
        _balances[from] = fromBalance - _taxAmount;
    }
    _balances[taxReciever] += _taxAmount;
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply. It is recommended to consolidate the deduction of both the `_receiverAmount` and the `_taxAmount` into a single operation to correctly reflect the total amount to be deducted from the sender's balance. This can be achieved by calculating the total deduction amount beforehand and then applying this total deduction in a single unchecked block. This approach ensures that the sender's balance is accurately updated in one step, preventing the current issue where the balance is incorrectly set twice.

IBD - Inefficient Balance Deduction

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L550
Status	Unresolved

Description

The contract calculates the `_receiverAmount` and the `_taxAmount` which are deducted from the `_balances` of the sender. However, the contract employs two separate `unchecked` blocks to deduct these amounts from the sender's balance. This approach not only increases the complexity of the code but also poses a risk of logical errors or inconsistencies in balance management. Specifically, the first unchecked block deducts the `_receiverAmount` from the sender's balance, and subsequently, another unchecked block deducts the `_taxAmount`. This method of deducting amounts in two steps could be streamlined for efficiency and clarity.

```
uint256 _taxAmount = (amount * _txTax) / 1000;
uint256 _autoburnAmount = (amount * _autoburnTax) / 1000;
unchecked {
    _balances[from] = fromBalance - _recieverAmount;
    }
    _balances[to] += _recieverAmount;
emit Transfer(from, to, _recieverAmount);
unchecked {
    _balances[from] = fromBalance - _taxAmount;
   }
```

Recommendation

It is recommended to consolidate the deduction process by assigning the sum of `_receiverAmount` and `_taxAmount` to a single variable, and then deducting this total amount from the sender's balance in one unchecked block. This approach simplifies the balance deduction process, reducing the risk of errors and improving the contract's readability and maintainability. By doing this, the code will only utilize one unchecked segment to accurately deduct the correct amount from the sender, enhancing the overall efficiency and safety of the contract's operations.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L466,481
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

The contract is utilizing the `changeTxTax` function that enables the owner to modify the transaction tax rates. This function includes a validation step to ensure that the sum of the new taxs does not exceed a predefined maximum limit (`MAXTAX`), which is internally set to `20` . However, the error message provided within the `require` statement misleadingly states that `you can't increase taxes above 5%` , despite the actual calculation being based on a maximum combined tax rate of 20 units. Given that the `MAXTAX` value represents a percentage in a base of `1000` , the maximum tax rate that can be set is effectively `2%` , not the `5%` as indicated by the error message. This discrepancy between the error message and the actual maximum tax rate can lead to confusion and misinterpretation of the contract's tax setting limitations.

```
   uint8 internal MAXTAX = 20;

    function changeTxTax(uint8 newTxTax, uint8 newAutoburnTax) public
onlyOwner {
        require(
            newTxTax + newAutoburnTax <= MAXTAX,
            "you can't increase taxes above 5%"
        );
    ...
    }
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract. It is recommended to modify the error message within the `changeTxTax` function to accurately reflect the maximum tax rate that can be applied. This adjustment will ensure clarity and prevent any misunderstanding regarding the tax rate modification capabilities of the contract, thereby enhancing the transparency and usability of the smart contract for its administrators.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L491
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 excludeFromTax(address user, bool exclude) public
onlyOwner {
        _isExcludedFromTax[user] = exclude;
        emit UserExcludedFromTax(user, exclude);
    }
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L466
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.

```
uint8 internal MAXTAX = 20
```

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	contracts/ShibaKeanu.sol#L463,464,466,467,668
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 internal _txTax = 0
uint8 internal _autoburnTax = 20
uint8 internal MAXTAX = 20
mapping(address => bool) internal _isExcludedFromTax

function SeeIfExcluded(
        address user
    ) public view returns (bool isExcludedFromTax) {
        return (_isExcludedFromTax[user]);
    }
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L501
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.

```
taxReciever = newReciever
```

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	contracts/ShibaKeanu.sol#L7,33,122,214,239,681,715
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 ^0.8.4;
```

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.

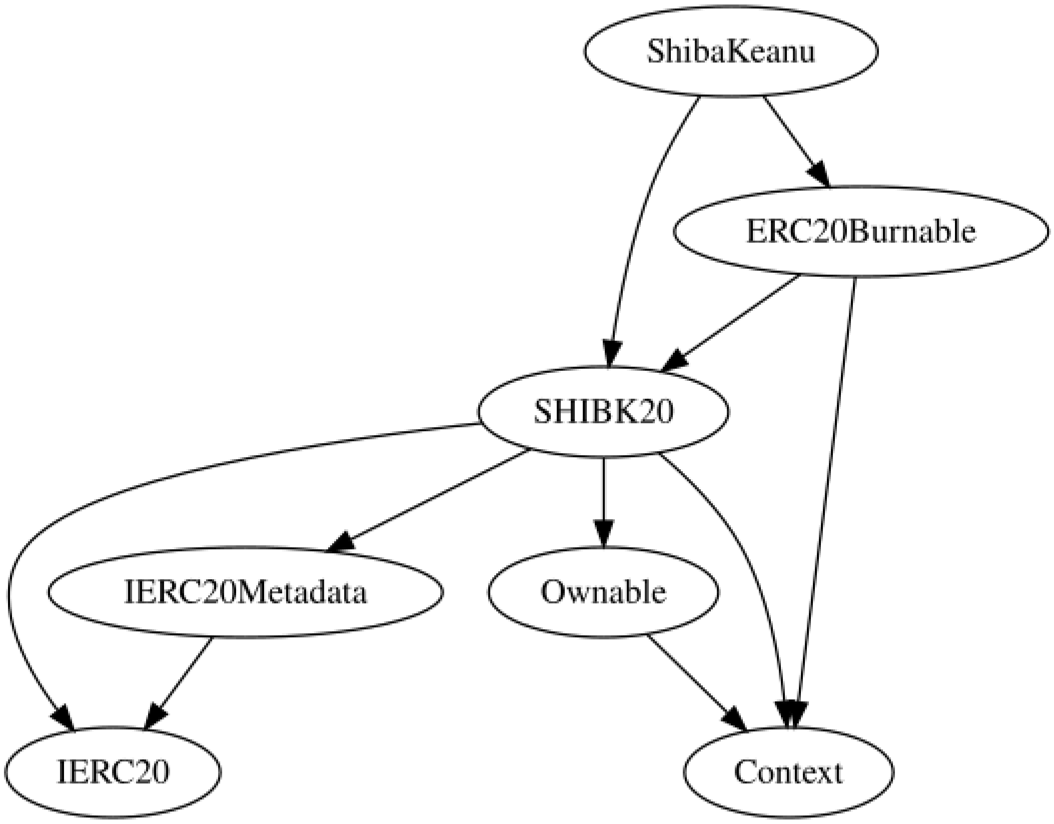
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

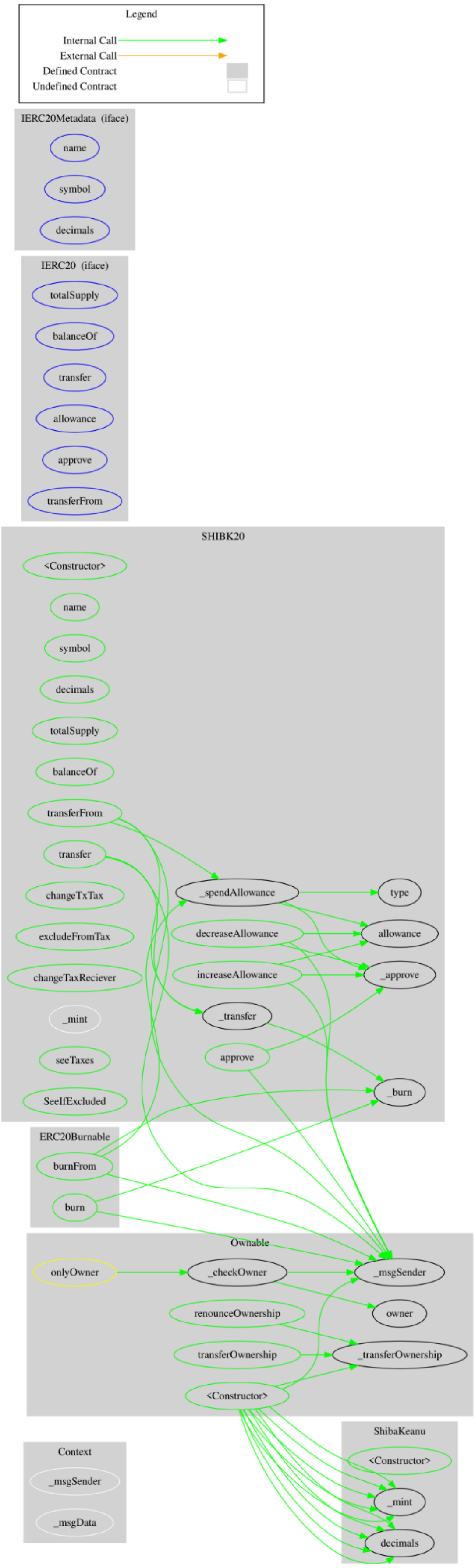
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
SHIBK20	Implementation	Context, IERC20, IERC20Metadata, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	changeTxTax	Public	✓	onlyOwner
	excludeFromTax	Public	✓	onlyOwner
	changeTaxReciever	Public	✓	onlyOwner

	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	seeTaxes	Public		-
	SeeIfExcluded	Public		-
ERC20Burnable	Implementation	Context, SHIBK20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
ShibaKeanu	Implementation	SHIBK20, ERC20Burnable		
		Public	✓	SHIBK20

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
SHIBK20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
changeTxTax
excludeFromTax
changeTaxReciever
_mint
seeTaxes
SeeIfExcluded
_spendAllowance
decreaseAllowance
increaseAllowance
_transfer
approve
type
allowance
_approve
_burn
ERC20Burnable
burnFrom
burn
Ownable
onlyOwner
_checkOwner
_msgSender
owner
renounceOwnership
transferOwnership
_transferOwnership
<Constructor>
Context
_msgSender
_msgData
ShibaKeanu
<Constructor>
_mint
decimals

Summary

ShibaKeanu contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. ShibaKeanu 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 2% fees.

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