Cyberscope

Audit Report

ShibaKeanu

February 2024

SHA256 71486524d50690950bb8228cfd588900f33dfaed8f3b3cae77e21695b5f0c8b2

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
Critical	TSD	Total Supply Diversion	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	GO	Gas Optimization	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L18	Multiple Pragma Directives	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	ShibaKeanu
Testing Deploy	https://testnet.bscscan.com/address/0x8f6ccab14729ba4739b685a7c36c7498d3b7d78d
Symbol	SHIBK
Decimals	18
Total Supply	888,000,000,000,000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	14 Feb 2024
Corrected Phase 2	18 Feb 2024

Source Files

Filename	SHA256
contracts/ShibaKeanu.sol	71486524d50690950bb8228cfd588900f33dfaed8f3b3cae77e21695b5f0c8b2

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	5	0	0	0

TSD - Total Supply Diversion

Criticality	Critical
Location	contracts/ShibaKeanu.sol#L559
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.

The following example is provided to further understand how the sum of balances is diverse from the total supply. Assume the contract has the following state:

Total supply: 10000

Account	Balance
0x123	2000
0x456	1000
taxReciever	0
owner	7000

- User 0x123 transfers 1000 tokens to user 0x456.
- User 0x123 balance is deducted by that amount (2000 - 1000 = 1000)
- `_taxAmount` = 1000 * 10 / 1000 = 10
- `_autoburnAmount` = 1000 * 10 / 1000 = 10

- `_recieverAmount` = 1000 - (10 + 10) = 980
- User 0x456 balance is increased by `_recieverAmount` (1000 + 980 = 1980)
- taxReciever balance is increased by `_taxAmount` (0 + 10 = 10)
- The `_autoburnAmount` is burned from the User 0x123 balance and total supply
    - User 0x123 balance is deducted by `_autoburnAmount` (1000 - 10 = 990)
    - Total supply is deducted by `_autoburnAmount` (10000 - 10 = 9990)

Summing the balances after transfer: `990 + 1980 + 10 + 7000 = 9980 != 9990`

```
unchecked {
    _balances[from] = fromBalance - amount;
}
uint256 _taxAmount = (amount * txTax) / 1000;
uint256 _autoburnAmount = (amount * autoburnTax) / 1000;
uint256 _recieverAmount = amount - (_taxAmount + _autoburnAmount);

_balances[to] += _recieverAmount;
emit Transfer(from, to, _recieverAmount);
_balances[taxReciever] += _taxAmount;
emit Transfer(from, taxReciever, _taxAmount);
_burn(from, _autoburnAmount);
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L535,542
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

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

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

GO - Gas Optimization

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L491
Status	Unresolved

Description

Gas optimization refers to the process of reducing the amount of gas required to execute a transaction. Gas is the unit of measurement used to calculate the fees paid to miners for including a transaction in a block on the blockchain.

The contract modifies the state of certain variables when the provided argument is different than their current state. However, in the case where the argument matches the currect state of the variable, the contract will not modify the state but the caller of the function will still be charged with gas.

```
function excludeFromTax(address user, bool exclude) public onlyOwner {
    if (isExcludedFromTax[user] != exclude){
        isExcludedFromTax[user] = exclude;}
    emit UserExcludedFromTax(user, exclude);
}
```

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.

The contract could modify these segments to use the `require` function provided by Solidity. By doing so, if the condition is not met, the transaction will revert and the caller will not be charged for executing the function.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L667
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 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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/ShibaKeanu.sol#L7,33,122,214,239,680,714
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 0.8.4;
pragma solidity ^0.8.4;
```

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	contracts/ShibaKeanu.sol#L7,33,122,214,239,680
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.

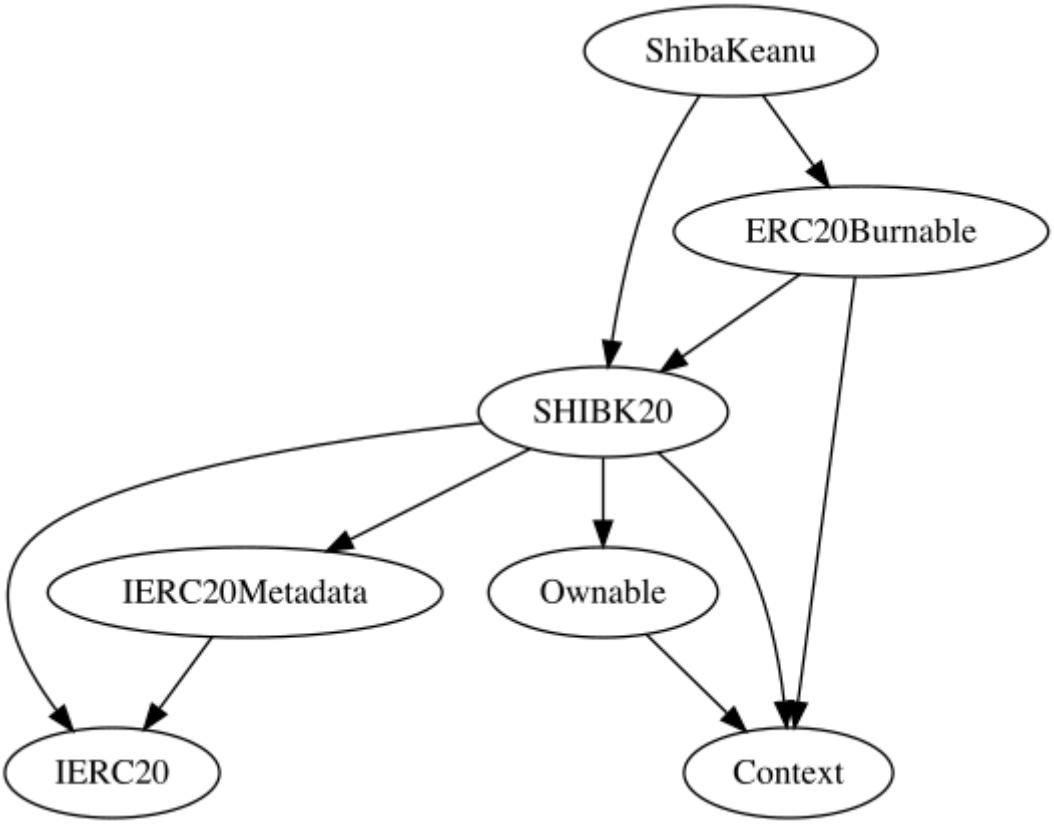
Functions Analysis

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	✓	-

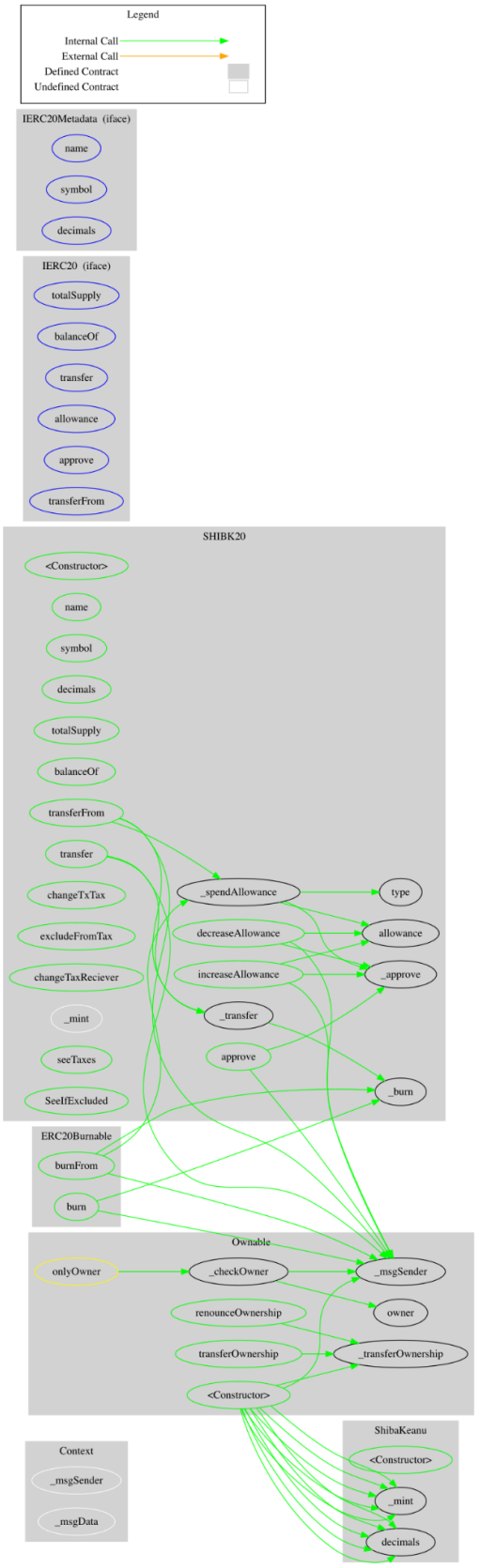
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
SHIBK20	Implementation	Context, IERC20, IERC20Metadata, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	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
renounceOwnership
transferOwnership
<Constructor>
_msgSender
owner
_transferOwnership
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 errors but one critical issue. 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