Cyberscope

Audit Report

DENJI INU

January 2023

Type	BEP20
Network	BSC
Address	0x59a1d6F2e3843B6d0cc618489a2836ff9AC716aE
Audited by	© cyberscope

Table of Contents

Review

Contract Name	DENJIINU
Compiler Version	v0.8.9+commit.e5eed63a
Optimization	200 runs
Explorer	https://bscscan.com/address/0x59a1d6f2e3843b6d0cc618489a2836ff9ac716ae
Address	0x59a1d6f2e3843b6d0cc618489a2836ff9ac716ae
Network	BSC
Symbol	DENJI
Decimals	9
Total Supply	1,000,000,000,000

Audit Updates

Initial Audit	03 Jan 2023

Source Files

Filename	SHA256
DENJIINU.sol	cf48917b5e111b895a502a24badae56f569d4c9a7914301e45fd5e9fa5c7e8d2

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
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L19	Stable Compiler Version	Unresolved

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	DENJIINU.sol#L71
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases unnecessarily the gas consumption.

```
library SafeMath {
    ...
}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	DENJIINU.sol#L37,181,182
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 _previousOwner
address payable private _developmentAddress =
payable(0x9fa2844B85B8Ae465ea2fd221BcD38Ca8FdfF966)
address payable private _marketingAddress =
payable(0x397613336a01090E64dD2c19d58f99B5e86086A8)
```

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	DENJIINU.sol#L133,156,157,158,165,180,191,192,193,404,559
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);
string private constant _name = "Denji Inu"
string private constant _symbol = "DENJI"
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 1000000000000 * 10**9
mapping (address => uint256) public _buyMap
uint256 public _maxTxAmount = 20000000000 * 10**9
uint256 public _maxWalletSize = 20000000000 * 10**9
uint256 public _swapTokensAtAmount = 100000 * 10**9
bool _tradingOpen
bool _swapEnabled
```

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	DENJIINU.sol#L37,161
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.

```
address private _previousOwner
mapping(address => uint256) private _tOwned
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	DENJIINU.sol#L547,555,565,569
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.

```
_redisFeeOnBuy = redisFeeOnBuy
_swapTokensAtAmount = swapTokensAtAmount
_maxTxAmount = maxTxAmount
_maxWalletSize = maxWalletSize
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	DENJIINU.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 ^0.8.9;
```

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.

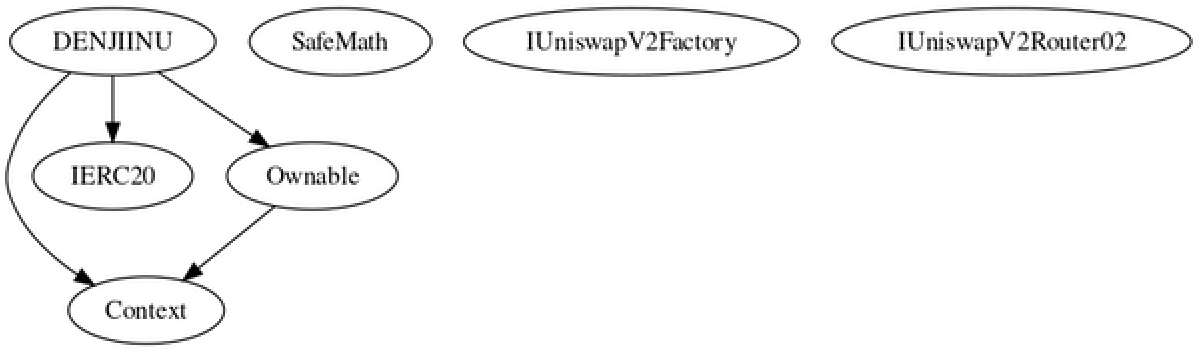
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		

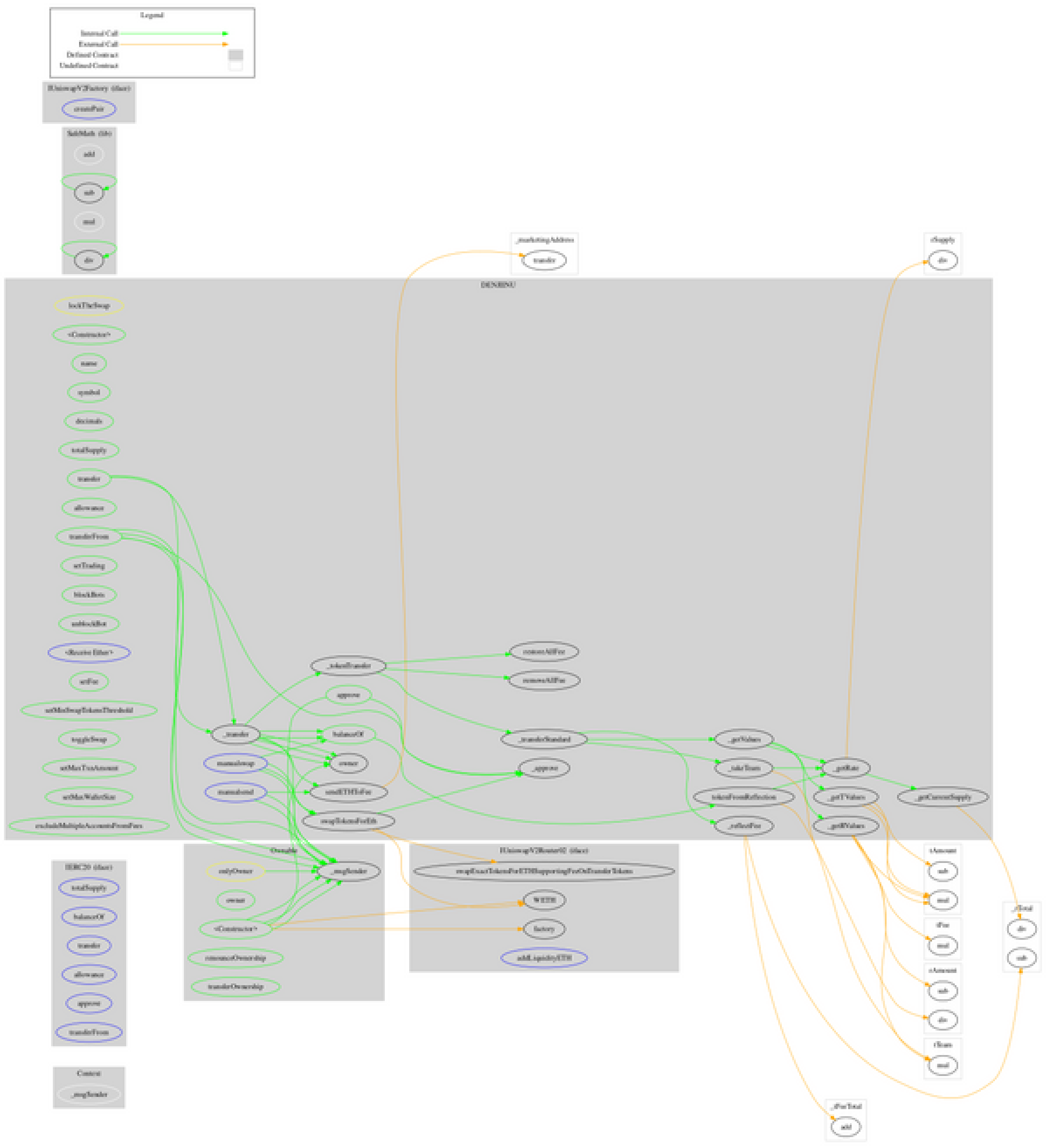
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
DENJIINU	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	tokenFromReflection	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	_approve	Private	✓	
	_transfer	Private	✓	
	swapTokensForEth	Private	✓	lockTheSwap

	sendETHToFee	Private	✓	
	setTrading	Public	✓	onlyOwner
	manualswap	External	✓	-
	manualsend	External	✓	-
	blockBots	Public	✓	onlyOwner
	unblockBot	Public	✓	onlyOwner
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_takeTeam	Private	✓	
	_reflectFee	Private	✓	
		External	Payable	-
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	setFee	Public	✓	onlyOwner
	setMinSwapTokensThreshold	Public	✓	onlyOwner
	toggleSwap	Public	✓	onlyOwner
	setMaxTxnAmount	Public	✓	onlyOwner
	setMaxWalletSize	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

DENJI INU 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 8% buy/sell tax fees and 1% buy/sell redis fees.

The owner has renounced ownership in the following transaction: https://bscscan.com/tx/0xb6ee5ff925df0606ab6f2ef0869848c32d39a86aeed4e18053ff5d7b0022a2da, therefore he has no access to admin function anymore and fees are locked in the percentages mentioned above.

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