Cyberscope

Audit Report

BADCAT

June 2024

Network	BASE
Address	0x1B2C141479757b8643A519Be4692904088d860B2
Audited by	© cyberscope

Table of Contents

Review

Contract Name	BADCAT
Compiler Version	v0.8.21+commit.d9974bed
Optimization	200 runs
Explorer	https://basescan.org/address/0x1b2c141479757b8643a519be4692904088d860b2
Address	0x1b2c141479757b8643a519be4692904088d860b2
Network	BASE
Symbol	BADCAT
Decimals	9
Total Supply	420,000,000,000

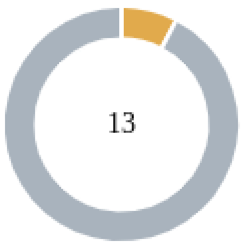
Audit Updates

Initial Audit	03 Jun 2024

Source Files

Filename	SHA256
BADCAT.sol	f86e0a9a5b86f39d6f1dd1baf3d66928da64a2a38122658feaf8afc159a2e623

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	12

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	12	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🟠	CSD	Circulating Supply Discrepancy	Unresolved
⚪	IDI	Immutable Declaration Improvement	Unresolved
⚪	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
⚪	RFC	Redundant Fee Calculation	Unresolved
⚪	RRA	Redundant Repeated Approvals	Unresolved
⚪	RSML	Redundant SafeMath Library	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L09	Dead Code Elimination	Unresolved
⚪	L13	Divide before Multiply Operation	Unresolved
⚪	L16	Validate Variable Setters	Unresolved
⚪	L17	Usage of Solidity Assembly	Unresolved

CSD - Circulating Supply Discrepancy

Criticality	Medium
Location	BADCAT.sol#L184
Status	Unresolved

Description

According to the ERC20 specification, the `totalSupply()` function should return the total supply of the token. The total supply should always equal the sum of the balances. The contract does not return the `totalSupply()`. Instead, the function returns the `totalSupply()` minus the amount that has been moved to the dead address. This amount is the circulating supply of the token. Many decentralized applications and tools are calculating many indicators like the circulating supply and market cap based on the `totalSupply()`. As a result, these applications will produce misleading results.

```
function totalSupply() public view override returns (uint256)
{return _totalSupply.sub(balanceOf(address(0)));}
```

Recommendation

The `totalSupply()` should always equal the sum of the holder's balances. The contract should comply with this convention so that the decentralized applications will produce correct results.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	BADCAT.sol#L152
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
pair
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	BADCAT.sol#L341
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
path[0] = address(this);
path[1] = router.WETH();
_approve(address(this), address(router), tokenAmount);
router.swapExactTokensForETHSupportingFeeOnTransferTokens(
    tokenAmount,
    0,
    path,
    address(this),
    block.timestamp);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RFC - Redundant Fee Calculation

Criticality	Minor / Informative
Location	BADCAT.sol#L210
Status	Unresolved

Description

The contract is designed to perform fee updates based on the time elapsed since `tradingStartTime`. However, after 15 minutes, the contract will always perform calculations leading to the final else condition where the fees are set to minimal values. This behavior occurs every time the `updateFees` function is called after the initial 15-minute period, resulting in unnecessary processing and potential inefficiencies.

```
 function updateFees() internal {
        if (block.timestamp < lastFeeUpdateTime + 1 minutes) {
            // If less than a minute has passed since the last update,
do nothing
            return;
        }

        // Update the lastFeeUpdateTime to the current time
        lastFeeUpdateTime = block.timestamp;

        uint256 timeElapsed = block.timestamp - tradingStartTime;
        if (timeElapsed < 1 minutes) {
           ...
        } else if (timeElapsed < 5 minutes) {
            ...
        } else if (timeElapsed < 10 minutes) {
            ...
        } else if (timeElapsed < 15 minutes) {
            ...
        } else {
            liquidityFee = 0;
            marketingFee = 1;
            developmentFee = 0;
            totalFee = 0;
            sellFee = 0;
            transferFee = 0;
        }
    }
```

Recommendation

It is recommended to update the `lastFeeUpdateTime` to a fixed value, such as the maximum possible value for a minute, within the final `else` condition. This change will prevent the function from re-entering the same condition repeatedly, thereby optimizing the contract's performance and reducing redundant calculations.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	BADCAT.sol#L339
Status	Unresolved

Description

The contract is designed to approve token transfers within every swap iteration by calling the `_approve` function before each swap operation. This approach results in additional gas costs since the approval process is repeated for every swap, leading to inefficiencies and increased transaction expenses.

```
    function swapTokensForETH(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();
        _approve(address(this), address(router), tokenAmount);
        router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp);
    }
```

Recommendation

It is recommended to optimize the contract by approving the token transfer once, rather than before every swap operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	BADCAT.sol
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 to 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 gas consumption unnecessarily in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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	BADCAT.sol#L116,132,133
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.

```
uint256 private _totalSupply =  420000000000 * (10 **
_decimals)
uint256 private denominator = 10000
bool private swapEnabled = true
```

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	BADCAT.sol#L75,113,114,115,120,137,144,145,146,182,258,287,328,377
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 = 'BADCAT'
string private constant _symbol = 'BADCAT'
uint8 private constant _decimals = 9
mapping (address => uint256) _balances
uint256 public _minTokenAmount = ( _totalSupply * 10 ) / 100000
address internal development_receiver =
0x4236e6A50B7c093C5c7EFAE5B4CAAdF084156C15
address internal marketing_receiver =
0x4236e6A50B7c093C5c7EFAE5B4CAAdF084156C15
address internal liquidity_receiver =
0x4236e6A50B7c093C5c7EFAE5B4CAAdF084156C15
bool _enabled
address _address
uint256 _sell
uint256 _liquidity
uint256 _development

...
```

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	BADCAT.sol#L260,282,374,379
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 = _liquidity
_maxTxAmount = maxTxAmount * (10**_decimals)
_minTokenAmount = newMinTokenAmount * (10 ** _decimals)
swapThreshold = _swapThreshold * (10 ** _decimals)
```

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	BADCAT.sol#L181
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 isCont(address addr) internal view returns (bool)
{uint size; assembly { size := extcodesize(addr) } return size
> 0; }
```

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	BADCAT.sol#L268,269,270,319,320,322,384
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.

```
require(sellFee <= denominator.div(100).mul(3), "Sell fee
cannot exceed 3%")
```

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	BADCAT.sol#L64,288,289,290
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
liquidity_receiver = _liquidity_receiver
marketing_receiver = _marketing_receiver
development_receiver = _development_receiver
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BADCAT.sol#L181
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(addr) }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

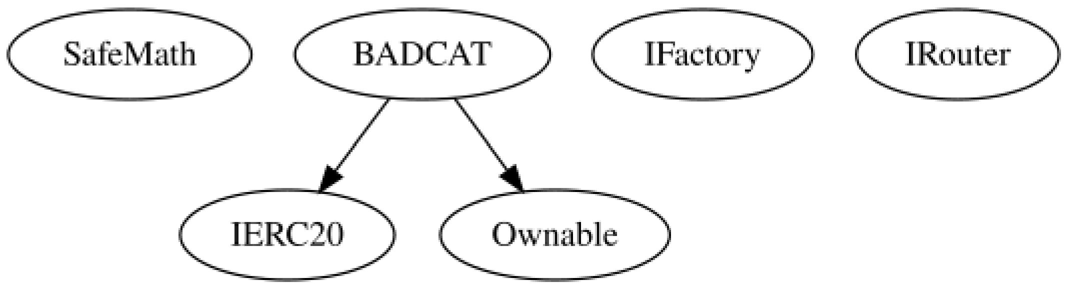
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
BADCAT	Implementation	IERC20, Ownable		
		Public	✓	Ownable
		External	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	enableTrading	External	✓	onlyOwner
	getOwner	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	isCont	Internal		
	setisfeeExempt	External	✓	onlyOwner
	approve	Public	✓	-
	totalSupply	Public		-
	getMaxWalletTokenAmount	Public		-
	getMaxTxAmount	Public		-

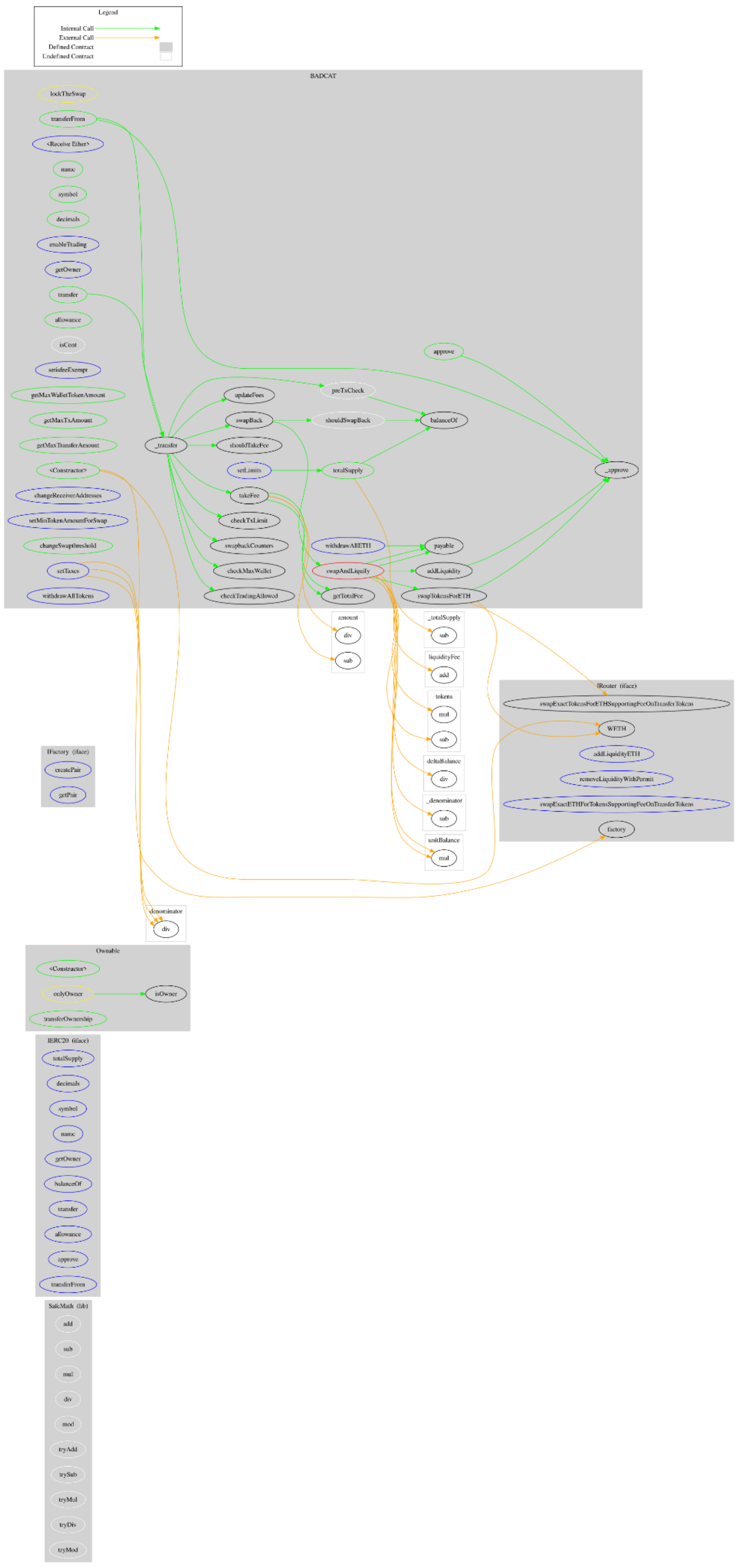
	getMaxTransferAmount	Public		-
	preTxCheck	Internal		
	_transfer	Private	✓	
	updateFees	Internal	✓	
	setTaxes	External	✓	onlyOwner
	setLimits	External	✓	onlyOwner
	changeReceiverAddresses	External	✓	onlyOwner
	checkTradingAllowed	Internal		
	checkMaxWallet	Internal		
	swapbackCounters	Internal	✓	
	checkTxLimit	Internal		
	swapAndLiquify	Private	✓	lockTheSwap
	addLiquidity	Private	✓	
	swapTokensForETH	Private	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	
	shouldTakeFee	Internal		
	getTotalFee	Internal		
	setMinTokenAmountForSwap	External	✓	onlyOwner
	changeSwapthreshold	Public	✓	onlyOwner
	takeFee	Internal	✓	
	transferFrom	Public	✓	-

	_approve	Private	✓	
	withdrawAllTokens	External	✓	onlyOwner
	withdrawAllETH	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BADCAT
lockTheSwap
transferFrom
<Receive Ether>
name
symbol
decimals
enableTrading
getOwner
transfer
allowance
isCont
setisfeeExempt
getMaxWalletTokenAmount
getMaxTxAmount
getMaxTransferAmount
<Constructor>
changeReceiverAddresses
setMinTokenAmountForSwap
changeSwapthreshold
setTaxes
withdrawAllTokens
_transfer
updateFees
swapBack
shouldTakeFee
setLimits
takeFee
checkTxLimit
swapbackCounters
checkMaxWallet
checkTradingAllowed
preTxCheck
shouldSwapBack
totalSupply
withdrawAllETH
swapAndLiquify
getTotalFee
approve
balanceOf
payable
addLiquidity
swapTokensForETH
_approve
amount
div
sub
_totalSupply
sub
liquidityFee
add
tokens
mul
sub
deltaBalance
div
_denominator
sub
unitBalance
mul
IRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
removeLiquidityWithPermit
swapExactETHForTokensSupportingFeeOnTransferTokens
factory
IFactory (iface)
createPair
getPair
denominator
div
Ownable
<Constructor>
onlyOwner
isOwner
transferOwnership
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
SafeMath (lib)
add
sub
mul
div
mod
tryAdd
trySub
tryMul
tryDiv
tryMod

Summary

BADCAT contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. BADCAT token is an interesting project that has a friendly and growing community.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

https://basescan.org/tx/0x09bc943c970269b68247d77f938b99ac0c73101274f571b60654bb632960826d

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