



Cyberscope

# Audit Report

## **BirbleAI**

June 2024

Network    BSC

Address    0x00c24E71f8a3468D5CD9368coF468C5b62055685

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
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSRS	Redundant SafeMath Require Statement	Unresolved
●	UMV	Unused Mintable Variable	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

# Table of Contents

<b>Analysis</b>	<b>1</b>
<b>Diagnostics</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>Review</b>	<b>5</b>
Audit Updates	5
Source Files	5
<b>Findings Breakdown</b>	<b>6</b>
IDI - Immutable Declaration Improvement	7
Description	7
Recommendation	7
MEM - Missing Error Messages	8
Description	8
Recommendation	8
RSML - Redundant SafeMath Library	9
Description	9
Recommendation	9
RSRS - Redundant SafeMath Require Statement	10
Description	10
Recommendation	10
UMV - Unused Mintable Variable	11
Description	11
Recommendation	11
L02 - State Variables could be Declared Constant	12
Description	12
Recommendation	12
L05 - Unused State Variable	13
Description	13
Recommendation	13
L09 - Dead Code Elimination	14
Description	14
Recommendation	14
L16 - Validate Variable Setters	16
Description	16
Recommendation	16
L19 - Stable Compiler Version	17
Description	17
Recommendation	17
<b>Functions Analysis</b>	<b>18</b>
<b>Inheritance Graph</b>	<b>19</b>

<b>Flow Graph</b>	<b>20</b>
<b>Summary</b>	<b>21</b>
<b>Disclaimer</b>	<b>22</b>
<b>About Cyberscope</b>	<b>23</b>

## Review

Contract Name	BirbleAIToken
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	<a href="https://bscscan.com/address/0x00c24e71f8a3468d5cd9368caf468c5b62055685">https://bscscan.com/address/0x00c24e71f8a3468d5cd9368caf468c5b62055685</a>
Address	0x00c24e71f8a3468d5cd9368caf468c5b62055685
Network	BSC
Symbol	BAITO
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

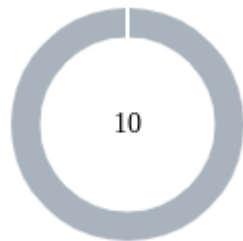
## Audit Updates

Initial Audit	18 Jun 2024
---------------	-------------

## Source Files

Filename	SHA256
BirbleAIToken.sol	985c3c756a584ecbefe24956a47dbf65cd9e0aee62bc5a040490610019adada0

## Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

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

## IDI - Immutable Declaration Improvement

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BirbleAIToken.sol#L807,808
<b>Status</b>	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.

```
mintable  
burnable
```

### 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.



## MEM - Missing Error Messages

Criticality	Minor / Informative
Location	BirbleAIToken.sol#L577
Status	Unresolved

### Description

The contract is missing error messages. Specifically, there are no error messages to 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.

```
require(  
    keccak256(abi.encodePacked(_addr)) ==  
    0x8e2ea2efa488794bc510dc250af50430af1f49e08f29a94eaf41a8b2f04cb  
    e06  
)
```

### 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.

## RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	BirbleAIToken.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>.

## RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	BirbleAIToken.sol#L150
Status	Unresolved

### Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");
    return c;
}
```

### Recommendation

It is recommended to remove the `require` statement from the `add` function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

## UMV - Unused Mintable Variable

Criticality	Minor / Informative
Location	BirbleAIToken.sol#L792,807
Status	Unresolved

### Description

The contract declares the bool public `mintable` variable that is never utilized within the contract's logic. Despite being assigned a value in the constructor, the `mintable` variable does not influence any subsequent operations or functions, leading to potential confusion and inefficiencies.

```
bool public mintable;
bool public burnable;

constructor(
    string memory name_,
    string memory symbol_,
    uint256 supply_,
    uint8 decimals_,
    bool canMint_,
    bool canBurn_,
    address addr_
) payable ERC20(name_, symbol_, decimals_, addr_) {
    ...
    mintable = canMint_;
    ...
}
```

### Recommendation

It is recommended to either remove the unused `mintable` variable if it is unnecessary or implement appropriate logic that leverages this variable to ensure that the contract functions as intended and maintains clarity.

## L02 - State Variables could be Declared Constant

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BirbleAIToken.sol#L790
<b>Status</b>	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 public Optimization = 171676162218129833
```

### 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.

## L05 - Unused State Variable

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BirbleAIToken.sol#L300
<b>Status</b>	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.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

### 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.

## L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	BirbleAIToken.sol#L346,351,362,730
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 abs(int256 a) internal pure returns (int256) {
    require(a != MIN_INT256);
    return a < 0 ? -a : a;
}

function toUint256Safe(int256 a) internal pure returns
(uint256) {
    ...
}

function toInt256Safe(uint256 a) internal pure returns (int256)
{
    int256 b = int256(a);
    require(b >= 0);
    return b;
}

...
```

### 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.



## L16 - Validate Variable Setters

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BirbleAIToken.sol#L484,805
<b>Status</b>	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.

```
_addr = addr_  
payable(addr_).transfer(msg.value)
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BirbleAIToken.sol#L6
<b>Status</b>	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 ^0.8.15;
```

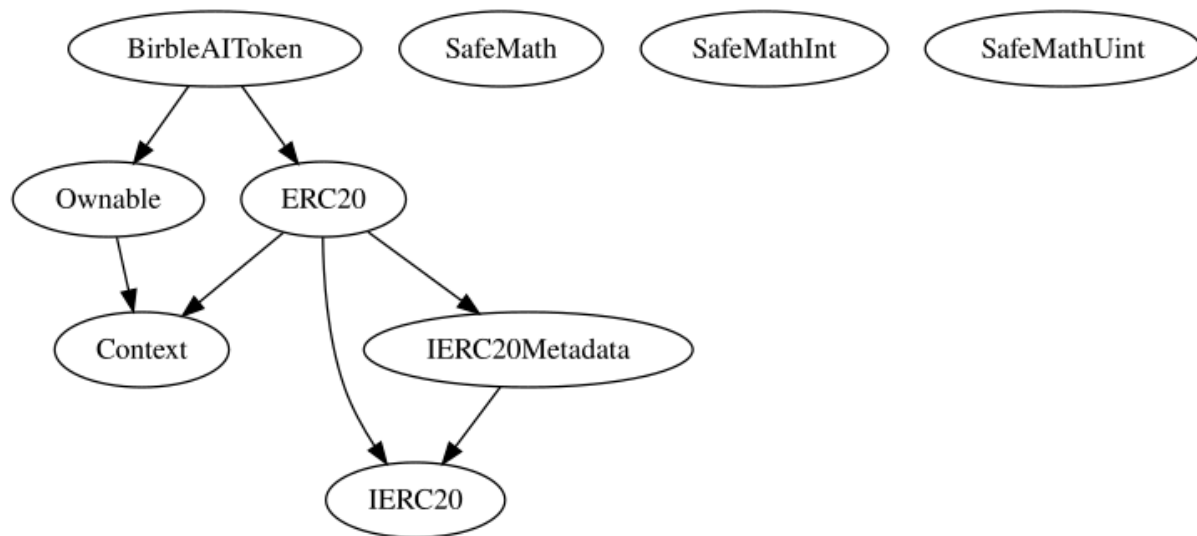
### Recommendation

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.

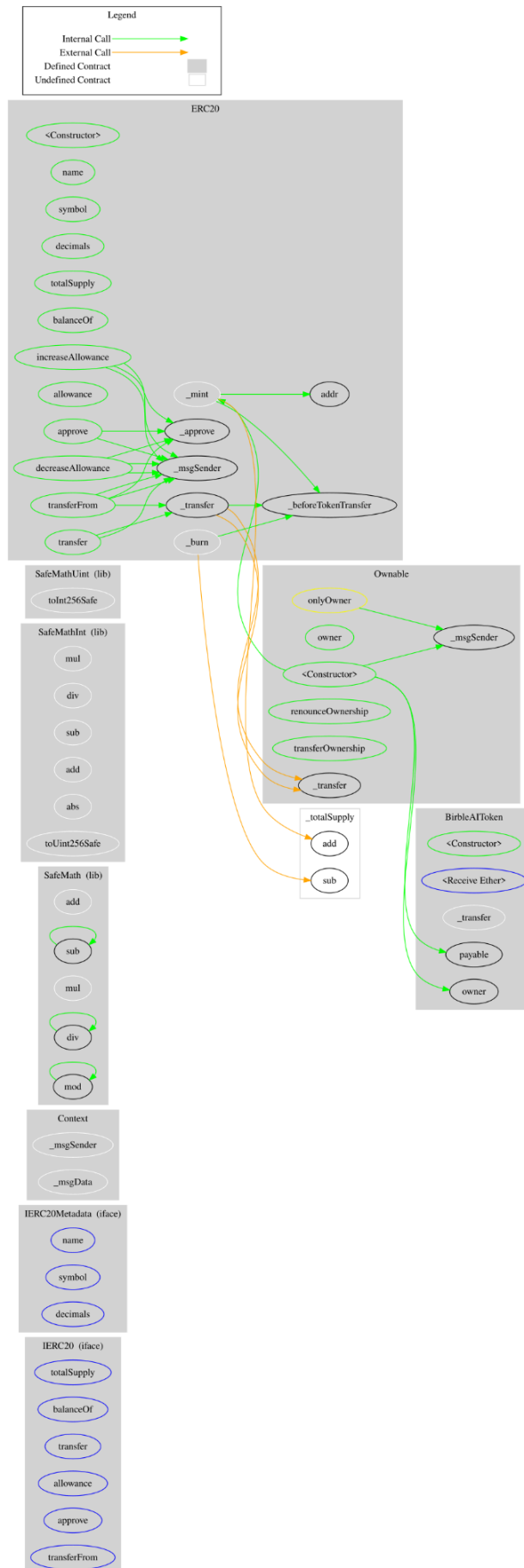
## Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
BirbleAIToken	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
		External	Payable	-
	_transfer	Internal	✓	

## Inheritance Graph



# Flow Graph



## Summary

BirbleAI contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. BirbleAI 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.

## 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>