Cyberscope

Audit Report

Chat AI

March 2023

Type	BEP20
Network	BSC
Address	0xa89bf95c5f15a847c8eb8d348cd7fed719ad7d80
Audited by	© cyberscope

Table of Contents

Review

Contract Name	AIToken
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0xa89bf95c5f15a847c8eb8d348cd7fed719ad7d80
Address	0xa89bf95c5f15a847c8eb8d348cd7fed719ad7d80
Network	BSC
Symbol	AI
Decimals	18
Total Supply	100,000,000

Audit Updates

Initial Audit	04 Mar 2023

Source Files

Filename	SHA256
BEP20.sol	9fbfcd45afb823a969893c25c91da344039d552cc9216b5569461a36d5ce8610
BEP20Detailed.sol	3a4c8b6fc4293d6d6bab8abf758ce39ca5938c61302f0d5b4cc5e72286dde246
BEPContext.sol	f8cb38984aade219964b1da398a8f6239bb1757065fd4dca760ca8d9b91ba5bb
BEPOwnable.sol	5c3c6d4514fa6c803ae0e1f0d048b5e3bfad80c3555e81614e487b3d47741072
IBEP20.sol	5953f0c5ecfc71226a4f934219c72dcbb2ae999aa58131cda84d28fd93a9caa7
SafeMath.sol	8c756491c47588016b20e85f9c12ecb5555a0d5a37737b39f8de3b6de5ad98b1
Token.sol	2337658a7180a606bd45bf3a4f567333774ecd649a4a8ef5b439ac0cc0148bda

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
	RS	Redundant Statements	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L09	Dead Code Elimination	Unresolved
	L11	Unnecessary Boolean equality	Unresolved
	L16	Validate Variable Setters	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

RS - Redundant Statements

Criticality	Minor / Informative
Location	Token.sol#L52
Status	Unresolved

Description

The contract contains code segments with fixed numbers that will never be executed. Since the buy and transfer tax is always zero, the statements that are using them will always yield zero amounts.

```
buyTax = 0;
transferTax = 0;

...

if(liquidityPool[sender] == true) {
    //It's an LP Pair and it's a buy
    taxAmount = (amount * buyTax) / 100;
}

...

} else {
    taxAmount = (amount * transferTax) / 100;
}
```

Recommendation

The team is advised to take into consideration these segments 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Token.sol#L28,29,30
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
sellTax
buyTax
transferTax
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Token.sol#L10,20,21,22,23,42,47
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.

```
mapping(address => bool) public _isExcludedFromFee
event changeTax(uint8 _sellTax, uint8 _buyTax, uint8 _transferTax);
event changeLiquidityPoolStatus(address lpAddress, bool status);
event changeMarketingPool(address marketingPool);
event change_isExcludedFromFee(address _address, bool status);
bool _status
address _lpAddress
address _marketingPool
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Token.sol#L76
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 transferToAddressETH(address payable recipient, uint256 amount)
private {
        recipient.transfer(amount);
  }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	Token.sol#L55,58
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
liquidityPool[sender] == true
liquidityPool[receiver] == true
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Token.sol#L48
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.

```
marketingPool = _marketingPool
```

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	Token.sol#L3
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 ^0.8.0;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Token.sol#L39
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
BEP20(token).transfer(marketingPool, amount)
```

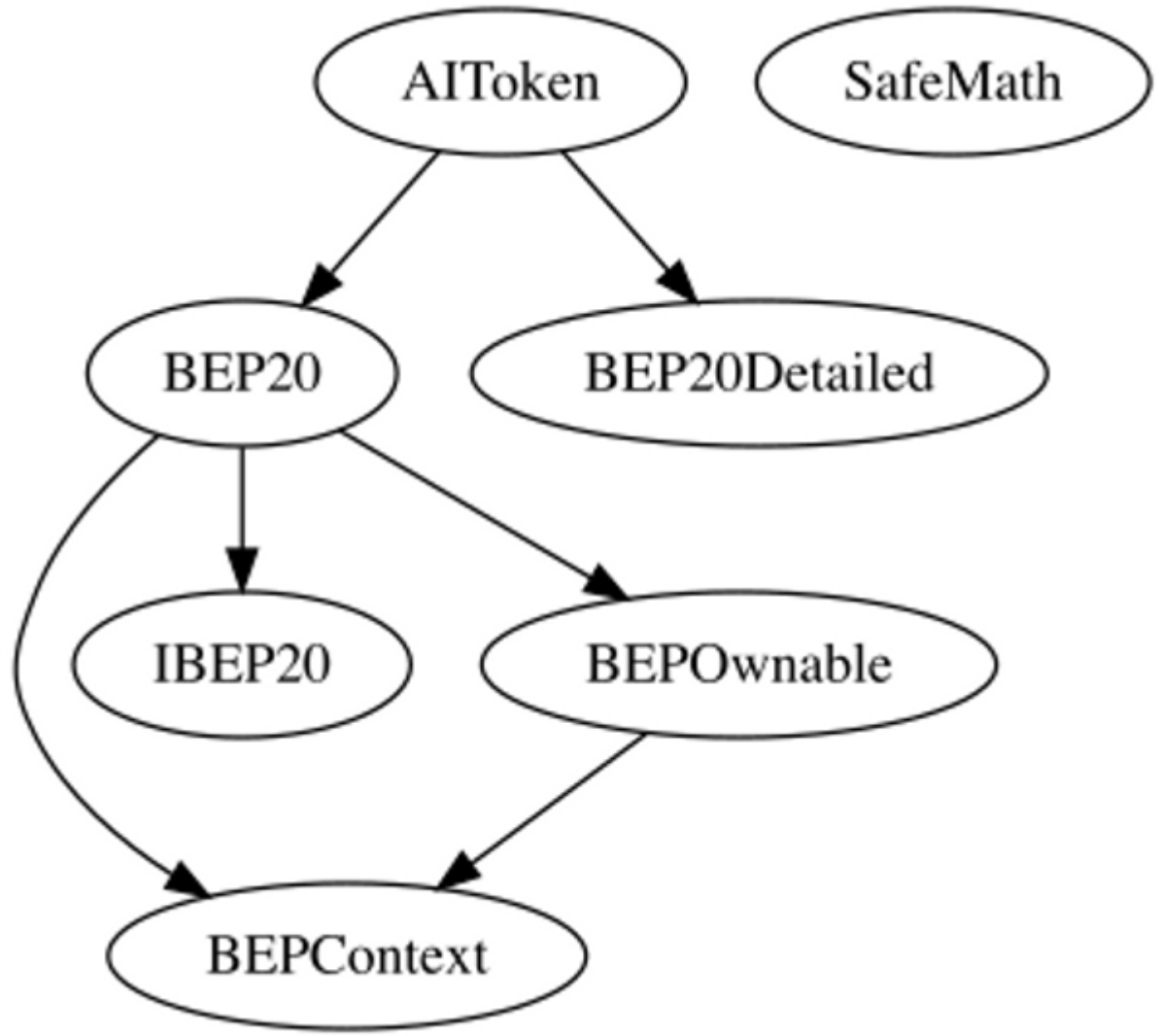
Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

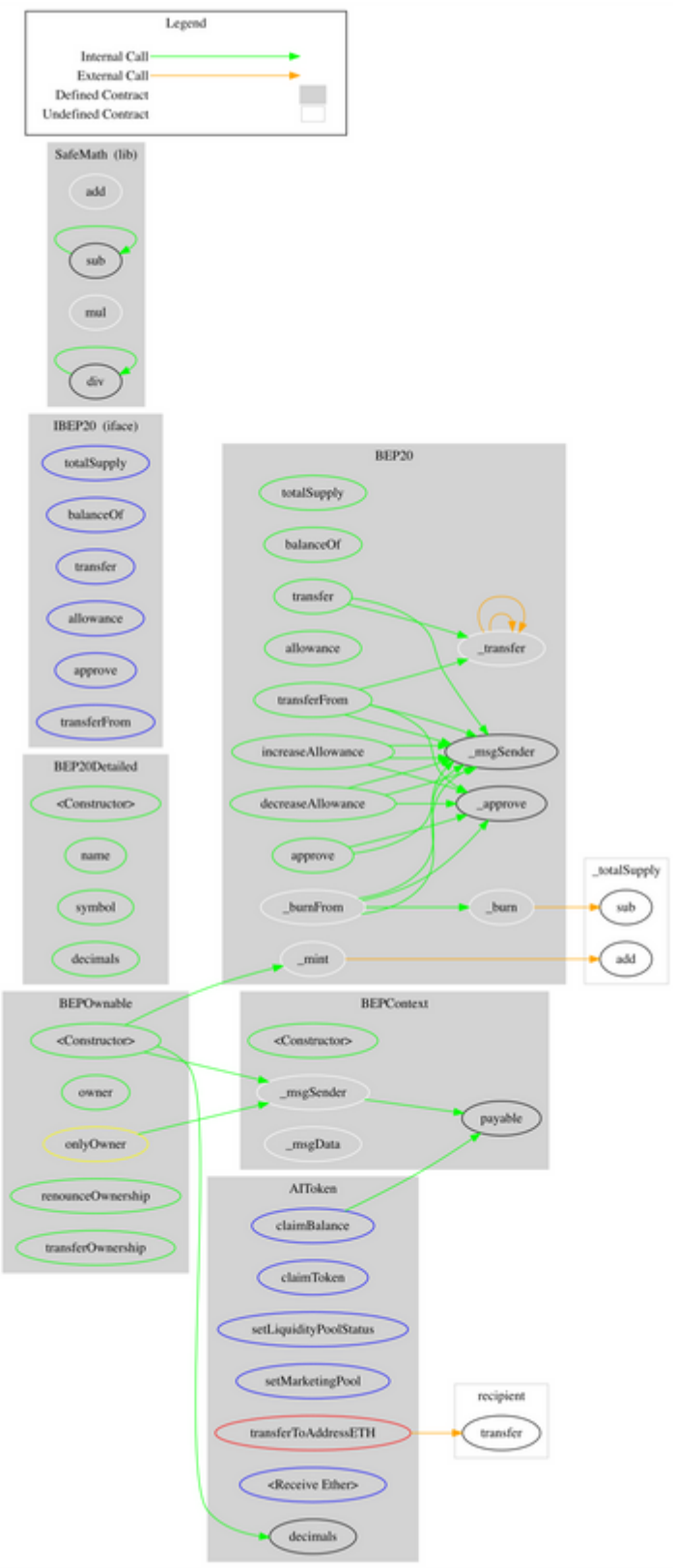
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AIToken	Implementation	BEP20Detailed, BEP20		
		Public	✓	BEP20Detailed
	claimBalance	External	✓	-
	claimToken	External	✓	-
	setLiquidityPoolStatus	External	✓	onlyOwner
	setMarketingPool	External	✓	onlyOwner
	_transfer	Internal	✓	
	transferToAddressETH	Private	✓	
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
mul
div
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
BEP20Detailed
<Constructor>
name
symbol
decimals
BEP20
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_burnFrom
_mint
_transfer
_msgSender
_approve
_burn
_totalSupply
sub
add
BEPOwnable
<Constructor>
owner
onlyOwner
renounceOwnership
transferOwnership
BEPContext
<Constructor>
_msgSender
_msgData
payable
AIToken
claimBalance
claimToken
setLiquidityPoolStatus
setMarketingPool
transferToAddressETH
<Receive Ether>
decimals
recipient
transfer

Summary

Chat AI contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Chat AI 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 1% fees in sales.

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