Cyberscope

Audit Report

AI SHIBA

January 2023

Type	BEP20
Network	BSC
Address	0x78C764eb414F29EF455d1905E4d75A050BA7e625
Audited by	© cyberscope

Table of Contents

Review

Contract Name	AiShiba
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	9999 runs
Explorer	https://bscscan.com/address/0x78c764eb414f29ef455d1905e4d75a050ba7e625
Address	0x78c764eb414f29ef455d1905e4d75a050ba7e625
Network	BSC
Symbol	AISHIBA
Decimals	18
Total Supply	100,000,000,000,000

Audit Updates

Initial Audit	20 Jan 2023 https://github.com/cyberscope-io/audits/blob/main/aishiba/v1/audit.pdf
Corrected Phase 2	23 Jan 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
contracts/AiShiba.sol	d95a861c134d22940333cab72d811d24f5b80dc7f23fa1ae0d1d21ae75f5e41f
contracts/AttributeMap.sol	71e8887c727be3dfd0c7d000d7d329f1e9bf4db56b7706b541504580391d46e8
contracts/Authorized.sol	3939842daf4a78df2c37cc0c7208a23f1b8e6a8c9b06e022415fe7adabed89a8
contracts/ERC20.sol	9a66f478ca1d9ab331c55603aba3ad649118be44de9830a21e7923083f667208
contracts/GasHelper.sol	410eaf449e756115d4d2f15cd1b191f42060caa5a1d26be37f8a80c1c3da8ab8
contracts/IPancake.sol	116d2bd2fdbb9dcca5b0bfe371e3d566f7cbb5e835b6238ac3ad7c16d25eed2d
contracts/SwapHelper.sol	e2b42eec42e51d0f52d749e8672921a0ed806fe9e27e364b332f0413bf68aab6

Solidity Assembly MethodId Analysis

MethodId	Method Name
0x70a08231	balanceOf(address)
0x022c0d9f	swap(uint256, uint256, address, bytes)
0x23b872dd	transferFrom(address, address, uint256)
0x0dfe1681	token0( )
0x0902f1ac	getReserves( )

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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	contracts/SwapHelper.sol#L15
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the team wallet. These funds have been accumulated from fees collected from the contract. The owner may take advantage of it by calling the `safeWithdraw` method.

```
function safeWithdraw() external onlyOwner {
  payable(_msgSender()).transfer(address(this).balance);
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped. Since a huge amount may volatile the token's price. The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RSC	Reflection Share Complexity	Unresolved
●	TSD	Total Supply Diversion	Unresolved
●	MBEI	Misleading Burn Event Issuer	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	CO	Code Optimization	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

●	L20	Succeeded Transfer Check	Unresolved

RSC - Reflection Share Complexity

Criticality	Minor / Informative
Status	Unresolved

Description

The contract is using a complex reward-sharing mechanism. On every transfer:

1. The sender's amount is tracked and potential reflections are minted to the sender's balance.
2. The reward mechanism is updated.

The mutation of the state variables, in combination with the complexity that is added to the contract, increases dramatically the gas consumption, the readability, and the maintainability of the contract. Lighter reflection mechanisms have been introduced, with significantly less complexity.

Recommendation

One of the initial well-known reflection approaches was introduced by the Safemoon project. The team is advised to reconsider the implementation of the reflection mechanism by investigating approaches like the Safemoon fork.

https://github.com/safemoonprotocol/Safemoon.sol/blob/main/Safemoon.sol

TSD - Total Supply Diversion

Criticality	Minor / Informative
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, if the variables `pausedSwapAdmin` or `pausedSwapPool` are disabled, then the amount is deducted from the sender but is not added to any address. On the other hand, if the variables `pausedSwapAdmin` or `pausedSwapPool` are disabled, then this amount will never be credited to an address. This makes the amount that is added to the total supply not equal to the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

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.

MBEI - Misleading Burn Event Issuer

Criticality	Minor / Informative
Location	contracts/AiShiba.sol#L347
Status	Unresolved

Description

As part of the taxing process, the contract burns an amount from the sender. The contract moves the amount to the contract's address and it burns it. This process generates a misleading event since the transfer to the zero address is issued from the contract address and not from the sender.

```
_balances[address(this)] += burnAmount;
_burn(address(this), burnAmount);
```

Recommendation

The team is advised to emit the burn transfer event by setting the sender as the issuer.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/AiShiba.sol#L189
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `_minAmountToAutoSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setMinAmountToAutoSwap(uint amount) public isAuthorized(1) {
    _minAmountToAutoSwap = amount;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/AiShiba.sol#L341
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
function splitFee(...) private {
    uint256 totalFee = adminFee + poolFee + burnFee + reflectFee;

    ...
    uint256 burnAmount = (incomingFeeAmount * burnFee) / totalFee;
    ...
    accumulatedToReflect += (incomingFeeAmount * reflectFee) / totalFee;
    ...
    accumulatedToAdmin += (incomingFeeAmount * adminFee) / totalFee;
    ...
    accumulatedToPool += (incomingFeeAmount * poolFee) / totalFee;
}
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/AiShiba.sol#L222,362,389,400,403
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Since the variable `_minAmountToAutoSwap` is a public variable of the contract and the argument `minTokenHolder` is always set to `_minAmountToAutoSwap`. The argument `uint minTokenHolder` is redundant.

```
uint public _minAmountToAutoSwap = 10000 * (10 ** decimals()); // 100

function _updateHolder(address holder, uint amount, uint minTokenHolder,
uint reflectPerShareValue) private

uint minTokenHolder = minTokenHoldToReflect;

_updateHolder(receiver, receiverAmount, minTokenHolder,
reflectPerShareValue);
```

Recommendation

It is recommended to remove redundant statements. 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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/AiShiba.sol#L51
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.

```
uint public totalBurned
```

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	contracts/AttributeMap.sol#L8 contracts/AiShiba.sol#L20,21,23,25,28,36,66
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 => uint) internal _attributeMap
string constant _name = "Ai Shiba"
string constant _symbol = "AISHIBA"
string public constant url = "www.aishiba.com"
uint constant maxSupply = 100_000_000_000_000e18
uint public _minAmountToAutoSwap = 10000 * (10 ** decimals())
uint constant maxTotalFee = 1000
uint private constant reflectPrecision = 10 ** 18
```

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	contracts/GasHelper.sol#L38,81
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 tokenTransfer(address token, address recipient, uint amount)
internal {
    bool failed = false;
    assembly {
      let emptyPointer := mload(0x40)
      mstore(emptyPointer,
0xa9059cbb0000000000000000000000000000000000000000000000000000000000000000)
      mstore(add(emptyPointer, 0x04), recipient)
      mstore(add(emptyPointer, 0x24), amount)
      failed := iszero(call(gas(), token, 0, emptyPointer, 0x44, 0, 0))
    }
    if (failed) revert("Unable to transfer token");
  }

...
```

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	contracts/AiShiba.sol#L289,301,313,317,326
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.

```
uint amountToSend = (amountOut * poolAmount) / (totalAmount)
uint amountBOptimal = (amountToSend * reserve1) / reserve0;
uint poolAmount = disabledAutoLiquidity ? accumulatedToPool :
accumulatedToPool
int amountAOptimal = (poolAmount * reserve0) / reserve1;/ 2
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/AiShiba.sol#L228,229,230,231
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint adminFee
uint poolFee
uint burnFee
uint reflectFee
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/AiShiba.sol#L163
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.

```
administrationWallet = account
```

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	contracts/GasHelper.sol#L27,40,53,67,83,96
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 {
      let emptyPointer := mload(0x40)
      mstore(emptyPointer,
0x0dfe168100000000000000000000000000000000000000000000000000000000)
      failed := iszero(staticcall(gas(), pair, emptyPointer, 0x04,
emptyPointer, 0x20))
      token0 := mload(emptyPointer)
    }

assembly {
      let emptyPointer := mload(0x40)
      mstore(emptyPointer,
0xa9059cbb00000000000000000000000000000000000000000000000000000000)
      mstore(add(emptyPointer, 0x04), recipient)
      mstore(add(emptyPointer, 0x24), amount)
      failed := iszero(call(gas(), token, 0, emptyPointer, 0x44, 0, 0))
    }

...
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/Authorized.sol#L33
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.

```
IERC20(token).transfer(receiver, 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**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		

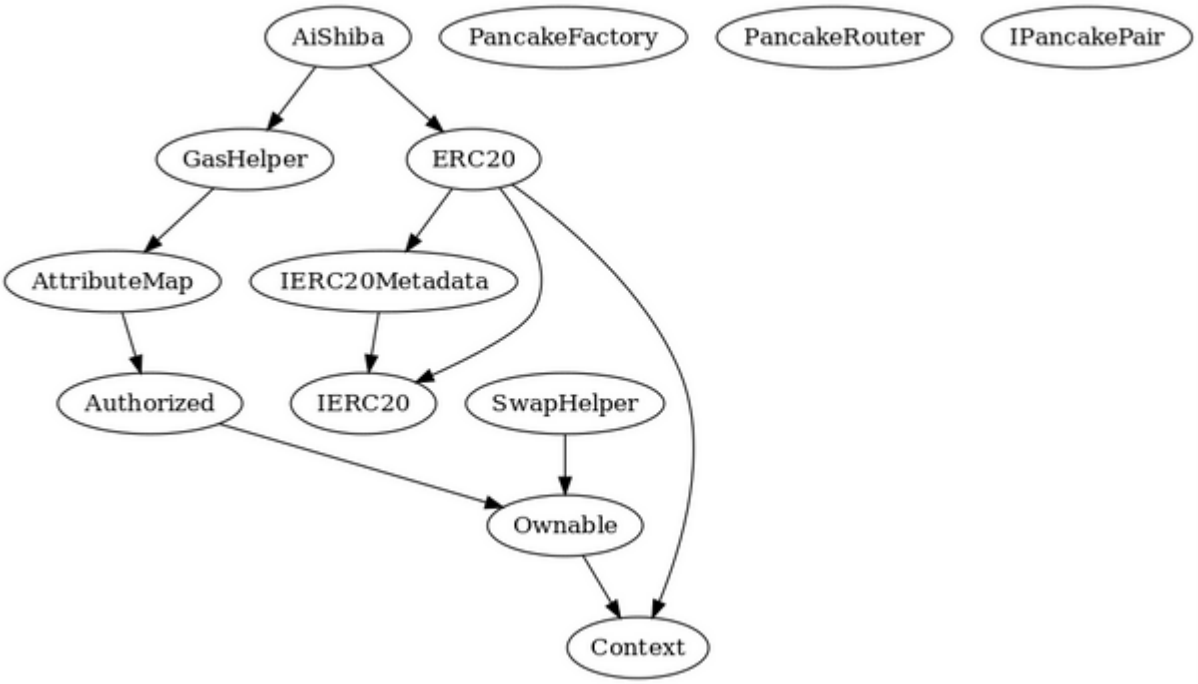
	_msgData	Internal		
AiShiba	Implementation	GasHelper, ERC20		
		External	Payable	-
		Public	✓	ERC20
	getOwner	External		-
	getFeeTotal	Public		-
	getSpecialWalletFee	Public		-
	balanceOf	Public		-
	setLiquidityPool	External	✓	isAuthorized
	setPausedSwapPool	External	✓	isAuthorized
	setPausedSwapAdmin	External	✓	isAuthorized
	setDisabledReflect	External	✓	isAuthorized
	setDisabledAutoLiquidity	External	✓	isAuthorized
	setAdministrationWallet	Public	✓	isAuthorized
	setSpecialWalletFeeOnSend	Public	✓	isAuthorized
	setSpecialWalletFeeOnReceive	Public	✓	isAuthorized
	setSpecialWalletFee	Private	✓	
	setMinAmountToAutoSwap	Public	✓	isAuthorized
	multiTransfer	External	✓	-
	burn	External	✓	-
	_transfer	Internal	✓	
	operateSwap	Private	✓	
	autoSwap	Private	✓	
	splitFee	Private	✓	
	setMinTokenHoldToReflect	External	✓	isAuthorized
	executeReflectOperations	Private	✓	
	_updateHolder	Private	✓	

AttributeMap	Implementation	Authorized		
	isExemptFeeSender	Public		-
	isExemptFeeReceiver	Public		-
	isExemptSwapperMaker	Public		-
	isExemptReflect	Public		-
	isSpecialFeeWalletSender	Public		-
	isSpecialFeeWalletReceiver	Public		-
	checkMapAttribute	Internal		
	isExemptFeeSender	Internal		
	isExemptFeeReceiver	Internal		
	isExemptSwapperMaker	Internal		
	isExemptReflect	Internal		
	isSpecialFeeWalletSender	Internal		
	isSpecialFeeWalletReceiver	Internal		
	setMapAttribute	Internal		
	applyMapAttribute	Internal		
	removeMapAttribute	Internal		
	setExemptFeeSender	Internal		
	setExemptFeeReceiver	Internal		
	setExemptSwapperMaker	Internal		
	setExemptReflect	Internal		
	setSpecialFeeWalletSender	Internal		
	setSpecialFeeWalletReceiver	Internal		
	setExemptFeeSender	Public	✓	isAuthorized
	setExemptFeeReceiver	Public	✓	isAuthorized
	setExemptSwapperMaker	Public	✓	isAuthorized
	setExemptReflect	Public	✓	isAuthorized
	setSpecialFeeWalletSender	Public	✓	isAuthorized
	setSpecialFeeWalletReceiver	Public	✓	isAuthorized

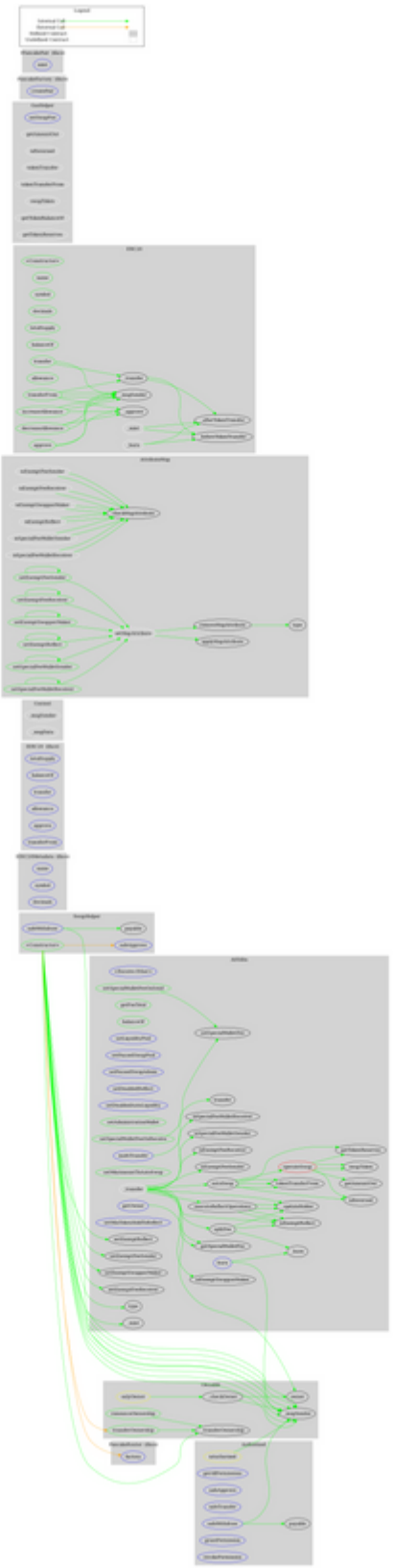
Authorized	Implementation	Ownable		
		Public	✓	-
	getAllPermissions	External		-
	safeApprove	External	✓	isAuthorized
	safeTransfer	External	✓	isAuthorized
	safeWithdraw	External	✓	isAuthorized
	grantPermission	External	✓	isAuthorized
	revokePermission	External	✓	isAuthorized
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	

	_afterTokenTransfer	Internal	✓	
GasHelper	Implementation	AttributeMap		
	setSwapFee	External	✓	isAuthorized
	getAmountOut	Internal		
	isReversed	Internal		
	tokenTransfer	Internal	✓	
	tokenTransferFrom	Internal	✓	
	swapToken	Internal	✓	
	getTokenBalanceOf	Internal		
	getTokenReserves	Internal		
PancakeFactory	Interface			
	createPair	External	✓	-
PancakeRouter	Interface			
	factory	External		-
IPancakePair	Interface			
	mint	External	✓	-
SwapHelper	Implementation	Ownable		
		Public	✓	-
	safeApprove	External	✓	onlyOwner
	safeWithdraw	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

AI SHIBA implements a token mechanism enriched with features like autogenerated-liquidity pool, reflections, and burning. This audit investigates security issues, business logic concerns, and potential improvements.

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