Cyberscope

Audit Report

AIGPROJECT

June 2023

SHA256 66d1b8e5ba7204e4f2a8806eeda0312c6cffc101bde0409e3745d982e54996d0

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
●	ZD	Zero Division	Unresolved
●	PSU	Potential Subtraction Underflow	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	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

●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	AIG
Testing Deploy	https://testnet.bscscan.com/address/0xeb43967b47e6f5119d612a429ae3c99a54db4ca5
Symbol	AIG
Decimals	9
Total Supply	3,000,000,000

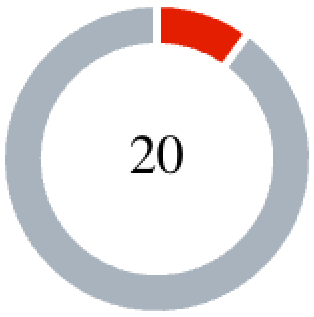
Audit Updates

Initial Audit	16 Jun 2023

Source Files

Filename	SHA256
contracts/testingDeploy/AIG.sol	5ed24d7c753f5862e8215ac48b2af81bf0543825b39d8f7996f47f2c7a6e09ed

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	18

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	18	0	0	0

ZD - Zero Division

Criticality	Critical
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

If the contract owner set the fees to zero (`totalBuyFee + totalSellFee`), then the swap will cause a zero division result.

```
uint256 swapTokens = tokens.mul(totalBuyFee + totalSellFee -
(sellFee.autoLP)/2)
    .div(totalBuyFee + totalSellFee);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

PSU - Potential Subtraction Underflow

Criticality	Critical
Location	contracts/testingDeploy/AIG.sol#L1732
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

The `swapAndLiquify` sends all the contract's ETH except the reward ratio to the Marketing address. Since the `swapAndLiquify` has liquidated the entire amount, then the amount before the swap will be greater than the amount after the swap. As a result the expression `address(this).balance - ethBalance` will revert.

```
uint256 ethBalance = address(this).balance;

swapAndLiquify(contractTokenBalance);
if(address(this).balance > 0){
    swapForPAXG(address(this).balance - ethBalance);
}
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	AIG.sol#L1999
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
if (address(this).balance - rewardPart > 0) {
    (bool sent, ) = marketingWallet.call{
        value: address(this).balance - rewardPart
    }("");
    require(sent, "eth transfer failed");
}
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	AIG.sol#L1989
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

The corresponding calculations `totalBuyFee + totalSellFee - (sellFee.autoLP)` take place multiple times.

```
uint256 lpPart = newBalance.mul((sellFee.autoLP) / 2).div(
    totalBuyFee + totalSellFee - (sellFee.autoLP) / 2
);
uint256 rewardPart = newBalance.mul(buyFee.reward + sellFee.reward).div(
    totalBuyFee + totalSellFee - (sellFee.autoLP) / 2
);
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	AIG.sol#L1988
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.

The variable `newBalance` might not be splitted as expected.

```
uint256 newBalance = address(this).balance.sub(initialBalance);
uint256 lpPart = newBalance.mul((sellFee.autoLP) / 2).div(
    totalBuyFee + totalSellFee - (sellFee.autoLP) / 2
);
uint256 rewardPart = newBalance.mul(buyFee.reward + sellFee.reward).div(
    totalBuyFee + totalSellFee - (sellFee.autoLP) / 2
);
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	AIG.sol#L1721
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` 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 setSwapTokens(uint256 amount) external onlyOwner {
    swapTokensAtAmount = amount * 1e9;
}
```

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.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	AIG.sol#L1707
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function setAutomatedMarketMakerPair(
    address pair,
    bool value
) public onlyOwner {
    require(
        pair != uniswapV2Pair,
        "AIG: The Uniswap pair cannot be removed from
automatedMarketMakerPairs"
    );

    _setAutomatedMarketMakerPair(pair, value);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	AIG.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.

```
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.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	AIG.sol#L248,253,263,269
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Map storage map
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	AIG.sol#L1496
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 deadWallet = address(0xdead)
```

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	AIG.sol#L9,1291,1368,1376,1385,1396,1633,1663,1664,1679,1692,1693,1694,2146
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);
uint256 internal constant magnitude = 2 ** 128
address _owner
address _token
address payable _marketing
address payable _liquidity
uint16 _reward
uint16 _marketing
uint16 _autoLP
address _account
```

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	AIG.sol#L368
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.

```
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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	AIG.sol#L1683,1700,1722,1966,1976
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.

```
totalBuyFee = buyFee.reward + buyFee.marketing
totalSellFee = sellFee.reward + sellFee.marketing + sellFee.autoLP
swapTokensAtAmount = amount * 1e9
maxTxAmount = amount
maxWallet = amount
```

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	AIG.sol#L248,414,1411
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 get(Map storage map, address key) internal view returns (uint)
{
        return map.values[key];
    }

function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

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	AIG.sol#L1989
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.

```
uint256 lpPart = newBalance.mul((sellFee.autoLP) / 2).div(
            totalBuyFee + totalSellFee - (sellFee.autoLP) / 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	AIG.sol#L1908,1943,1944,1945
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.

```
uint256 fees
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	AIG.sol#L1312,1313,1368,1376,1385,1396
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	AIG.sol#L1671
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.

```
lpWallet = _liquidity
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	AIG.sol#L1641
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.

```
erc20token.transfer(owner(), balance)
```

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**
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-

	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IterableMapping	Library			
	get	Internal		

	getIndexOfKey	Internal		
	getKeyAtIndex	Internal		
	size	Internal		
	set	Internal	✓	
	remove	Internal	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		

SafeMathUint	Library			
	toInt256Safe	Internal		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		

		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-

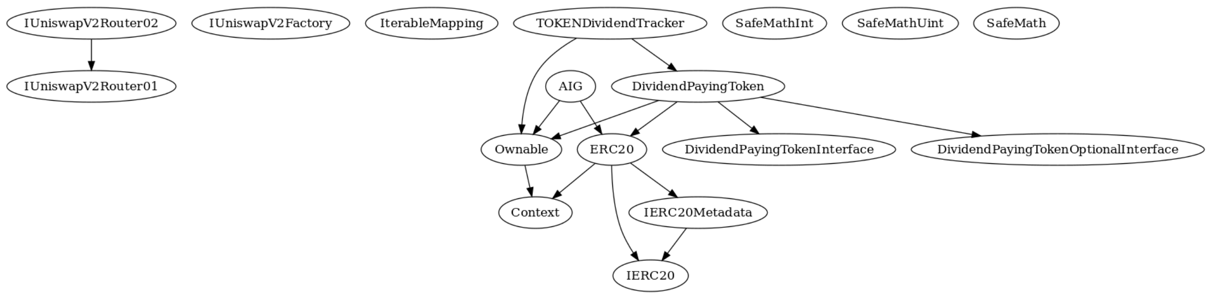
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
DividendPaying Token	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
	decimals	Public		-
	distributePAXGDividends	Public	✓	onlyOwner

	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
AIG	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	decimals	Public		-
	updateRouter	External	✓	onlyOwner
	claimStuckTokens	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	setSwapTokens	External	✓	onlyOwner

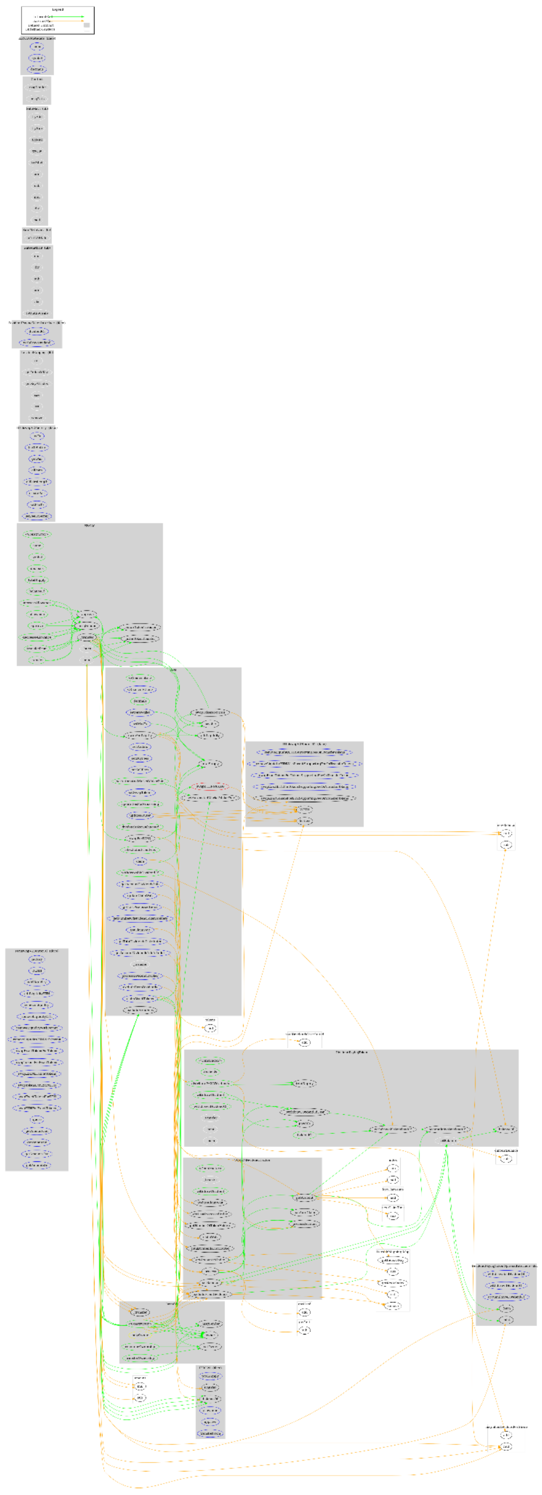
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	excludeFromDividends	External	✓	onlyOwner
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	_transfer	Internal	✓	
	setMaxTx	External	✓	onlyOwner
	setMaxWallet	External	✓	onlyOwner
	swapAndLiquify	Private	✓	
	addLiquidity	Private	✓	
	swapTokensForETH	Private	✓	
	swapETHForPAXG	Private	✓	
	swapForPAXG	Private	✓	

TOKENDividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPaying Token
	_transfer	Internal		
	withdrawDividend	Public		-
	excludeFromDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

AIGPROJECT contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The contract contains some misleading concepts that may cause critical issues. There is also a limit of max 10% fee.

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