Cyberscope

Audit Report

Arb Axolotl

May 2023

SHA256 d512ea32db6a862bb6ea815eff8cb729568e833478d7f314ff6db3f8c1662572

Audited by © cyberscope

Table of Contents

Review

Contract Name	Axolotl_AI
Testing Deploy	https://testnet.bscscan.com/address/0x69906f6c344f4f05614e604430efa96dc8514fee
Symbol	ARBAX
Decimals	6

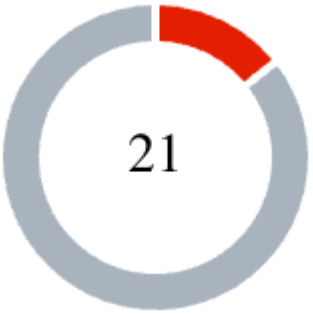
Audit Updates

Initial Audit	14 May 2023

Source Files

Filename	SHA256
contracts/testingDeploy/Contract.sol	d512ea32db6a862bb6ea815eff8cb729568e833478d7f314ff6db3f8c1662572

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	18	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
	ST	Stops Transactions	Unresolved
	OCTD	Transfers Contract's Tokens	Passed
	OTUT	Transfers User's Tokens	Passed
	ELFM	Exceeds Fees Limit	Unresolved
	ULTW	Transfers Liquidity to Team Wallet	Passed
	MT	Mints Tokens	Passed
	BT	Burns Tokens	Passed
	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L2060
Status	Unresolved

Description

As part of the launch process, initially, the transfers are disabled for all the users excluding the authorized addresses. Once the trades are enabled it will not be able to stop again.

```
if (!canAddLiquidityBeforeLaunch[sender]) { //If trading isn't
opened yet, and you are not specially authorized, you cant transfer
or trade tokens
    require(launched(), "Trading not open yet");
}
```

Recommendation

The contract could embody a check for not allowing setting the _maxTxAmount less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. 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. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/testingDeploy/Contract.sol#L2251
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setOverallFees` function with a high percentage value.

```
function setOverallFees(
    uint256 _holderFee,
    uint256 _liquidityFee,
    uint256 _flexFee,
    uint256 _marketingFee

) external onlyOwner {
    holderFee = _holderFee;
    liquidityFee = _liquidityFee;
    flexFee = _flexFee;
    marketingFee = _marketingFee;

    totalFee = _holderFee + _liquidityFee + _flexFee +
_marketingFee + devFee;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	IDE	Invalid Distributor Execution	Unresolved
●	ZD	Zero Division	Unresolved
●	RTCI	Reward Token Change Inconsistency	Unresolved
●	RMBS	Redundant Mint Burn Sequence	Unresolved
●	CR	Code Repetition	Unresolved
●	PAP	Pair Address Preexistence	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	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

●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

IDE - Invalid Distributor Execution

Criticality	Critical
Location	contracts/testingDeploy/Contract.sol#L2113
Status	Unresolved

Description

As part of the swap process, the contract sends the porpotional `holderFee` to the `dividendDistributor` address. The deposit method expects ETH but the amount is calculated based on the $ARBAX tokens. As a result, the contract will not have the funds to cover the `amountAxoHolder` amount and the transaction will revert.

```
try dividendDistributor.deposit{value: amountAxoHolder}() {} catch
{}
```

Recommendation

The team is adviced to revesite the business logic of the fees distribution system. The contract should deposit to the destributor address native ETH instead of tokens.

ZD - Zero Division

Criticality	Critical
Location	contracts/testingDeploy/Contract.sol#L2111
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.

```
uint256 amountAxoLp = (taxAmount * liquidityFee) / (totalFee);
uint256 amountAxoHolder = (taxAmount * holderFee) / (totalFee);
```

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 executing of the corresponding statements.

RTCI - Reward Token Change Inconsistency

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L2344
Status	Unresolved

Description

The contract owner has the authority to change the distribution token address by calling the `setRewardToken()` method. This may produce several issues in the distribution process.

1. There should be a valid pair address between the ETH and the new token address.
2. The internal state of the distributor should reset since variables like `dividendsPerShare` are based on the previous token's balance.

```
function setRewardToken(address _rewardToken) external onlyOwner {
    dividendDistributor.setRewardTokenInternally(_rewardToken);
}
```

Recommendation

The team is advised to either remove the option of changing the reward address or resolve the side-effects of the change.

RMBS - Redundant Mint Burn Sequence

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L2239
Status	Unresolved

Description

As part of the tokens distribution process, the $ARBAX address mints 20% of tokens, and then it burns the same amount. The balances and the total supply before and after this sequence is exactly the same. As a result, this execution is redundant.

```
_mint(address(this),  (_targettotalSupply * 20) / 100);
_burn(address(this), (_targettotalSupply * 20) / 100);
```

Recommendation

The team is adviced to remove the mint and burn sequence since it will produce the same result.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L2204
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

All of the three methods `clearStuckBalance` , `rescueToken` and `rescueArbax` are subset of the `rescueToken` method.

```
function clearStuckBalance() external onlyOwner {
    backToken.transfer(_msgSender(), backToken.balanceOf(address(this)));
}

function rescueToken(address tokenAddress) external onlyOwner {

IERC20(tokenAddress).safeTransfer(msg.sender,IERC20(tokenAddress).balanceOf(ad
dress(this)));
}

function rescueArbax() external onlyOwner {
    _transfer(address(this), msg.sender, this.balanceOf(address(this)));
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

Both `rescueArbax` and `clearStuckBalance` methods could be removed since the `rescueToken` method can produce the same result with the proper arguments.

PAP - Pair Address Preexistence

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L2033
Status	Unresolved

Description

The contract initializes the pait address in the `initializePair()` method. If a third user create the pair address prior the `initializePair()` , then this method will not be able to be called again since the `createPair()` will revert.

```
function initializePair() external onlyOwner {
    require(!initialized, "Already initialized");
    address pair = factory.createPair(address(WETH),
address(this));
    isDividendExempt[pair] = true;
    _pairs.add(pair);
    initialized = true;
}
```

Recommendation

The team is advised to move the pair creation in the constructor to guarantee that the pair will not exist. Otherwise, the team could exploit the `getPair()` method to check if the pair address already exists.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.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 on 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 the 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	contracts/testingDeploy/Contract.sol#L1194,1201,1215,1227,1260,1267, 1281,1293,1334,1341,1355,1367,1408,1415,1429,1441
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.

```
Set storage set
Bytes32Set storage set
AddressSet storage set
UintSet storage set
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L2013,2024,2025
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`.

```
backToken
router
dividendDistributor
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L1593,1605,1970,1972,1987,2000,2001
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 routerAddress = 0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
uint256 public devFee = 100
uint256 public feeDenominator = 10000
bool private launchburn
uint256 _targettotalSupply = 10_000_000_000 * 1e6
```

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/testingDeploy/Contract.sol#L626,1528,1584,1594,1721,1760,1944,1992,2001,2252,2253,2254,2255,2267,2268,2269,2270,2271,2272,2286,2290,2344
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function WETH() external pure returns (address);
address _token
IBEP20 RewardToken =
IBEP20(0x912CE59144191C1204E64559FE8253a0e49E6548)
address _rewardToken

...
```

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	contracts/testingDeploy/Contract.sol#L1987
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.

```
bool private launchburn
```

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	contracts/testingDeploy/Contract.sol#L1629,2258,2341
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.

```
minPeriod = newMinPeriod
holderFee = _holderFee
distributorGas = gas
```

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	contracts/testingDeploy/Contract.sol#L689,714,743,774,784,799,809,848,903,919,934,943,956,1227,1253,1260,1267,1281,1293,1367,1401,1408,1415,1429,1441
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient
may have reverted");
    }

function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

...
```

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

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L2274,2275,2276,2277,2278,2279
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.

```
flexWallet = _flexWallet
marketingWallet = _marketingWallet
devWallet = _devWallet
airdropWallet = _airdropWallet
liqWallet = _liqWallet
presaleWallet = _presaleWallet
```

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/testingDeploy/Contract.sol#L865,1298,1372,1446
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 returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
            result := store
        }
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L5,85,128,156,182,571,633,879,995,1079,1457,1755,1857,1918,1940
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.1;
pragma solidity >=0.5.0;
pragma solidity >=0.6.2;
pragma solidity =0.8.19;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/testingDeploy/Contract.sol#L5,85,128,156,182,571,633,879,995,1079
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;
pragma solidity ^0.8.1;
```

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	contracts/testingDeploy/Contract.sol#L1705,1718,2145,2146,2147,2219
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.

```
RewardToken.transfer(shareholder, amount)
RewardToken.transfer(to, RewardToken.balanceOf(address(this)))
backToken.transfer(flexWallet, amountBackTokenFlex)
backToken.transfer(marketingWallet, amountBackTokenMarketing)
backToken.transfer(devWallet, amountBackTokenDev)
backToken.transfer(_msgSender(),
backToken.balanceOf(address(this)))
```

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.

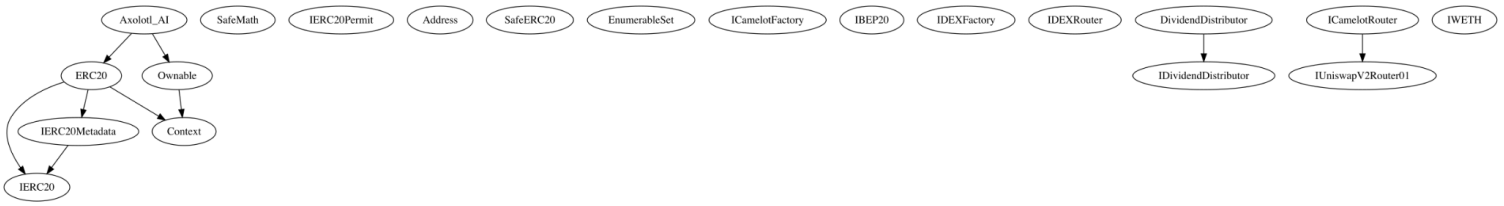
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	onlyToken
	rescueDividends	External	✓	onlyToken
	setRewardTokenInternally	External	✓	onlyToken
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
Axolotl_AI	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	initializePair	External	✓	onlyOwner

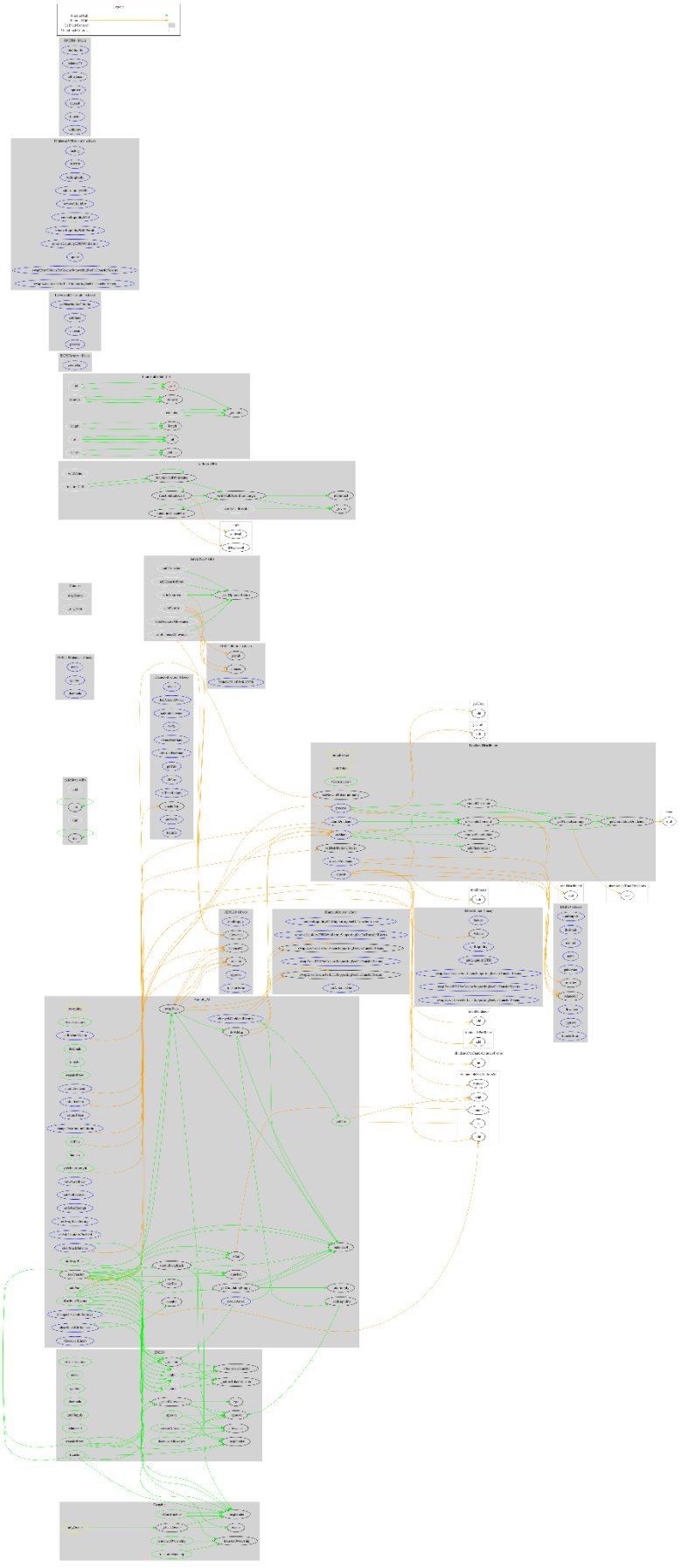
	decimals	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_axoTransfer	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	_doAddLp	Internal	✓	
	_addLiquidity	Internal	✓	
	doSwapBack	Public	✓	onlyOwner
	launched	Internal		
	takeFee	Internal	✓	
	rescueToken	External	✓	onlyOwner
	rescueArbax	External	✓	onlyOwner
	clearStuckEthBalance	External	✓	onlyOwner
	clearStuckBalance	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	launch	Public	✓	onlyOwner
	distributeTokens	Public	✓	onlyOwner
	setOverallFees	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	setIsFeeExempt	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	setAddLiquidityEnabled	External	✓	onlyOwner

	isPair	Public		-
	addPair	Public	✓	onlyOwner
	delPair	Public	✓	onlyOwner
	getMinterLength	Public		-
	getPair	Public		-
	claimDividend	External	✓	-
	changeIsDividendExempt	External	✓	onlyOwner
	changeDistributionCriteria	External	✓	onlyOwner
	changeDistributorSettings	External	✓	onlyOwner
	setRewardToken	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

Arb Axolotl contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions and manipulating the fees. The team is advised to reconsider segments of the business logic and revisit some parts of the implementation.

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