



Cyberscope

Audit Report

Kakapo Protocol

May 2023

Network BSC

Address 0x003882834E096c5a07dF4438D32cFe36BBdcE322

Audited by © cyberscope

Table of Contents

Table of Contents	1
Review	2
Audit Updates	2
Source Files	2
Findings Breakdown	3
Analysis	4
ST - Stops Transactions	5
Description	5
Recommendation	5
Diagnostics	6
PTRP - Potential Transfer Revert Propagation	7
Description	7
Recommendation	7
RSW - Redundant Storage Writes	8
Description	8
Recommendation	8
MSE - Missing Solidity Events	9
Description	9
Recommendation	9
RVD - Redundant Variable Declaration	10
Description	10
Recommendation	10
IDI - Immutable Declaration Improvement	11
Description	11
Recommendation	11
L02 - State Variables could be Declared Constant	12
Description	12
Recommendation	12
L04 - Conformance to Solidity Naming Conventions	13
Description	13
Recommendation	14
L05 - Unused State Variable	15
Description	15
Recommendation	15
L07 - Missing Events Arithmetic	16
Description	16
Recommendation	16
L13 - Divide before Multiply Operation	17
Description	17

Recommendation	17
L14 - Uninitialized Variables in Local Scope	18
Description	18
Recommendation	18
L19 - Stable Compiler Version	19
Description	19
Recommendation	19
L20 - Succeeded Transfer Check	20
Description	20
Recommendation	20
Functions Analysis	21
Inheritance Graph	25
Flow Graph	26
Summary	27
Disclaimer	28
About Cyberscope	29

Review

Contract Name	KakapoProtocol
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0x003882834e096c5a07df4438d32cfe36bbdce322
Address	0x003882834e096c5a07df4438d32cfe36bbdce322
Network	BSC
Symbol	Kakapo
Decimals	18
Total Supply	10,000,000

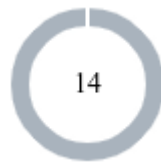
Audit Updates

Initial Audit	25 May 2023
---------------	-------------

Source Files

Filename	SHA256
KakapoProtocol.sol	cdeaf014d5a0adaee1bcfb6d826121326e23ca391c26b2af413d3f8940954a87

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	14	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	Passed
●	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	KakapoProtocol.sol#L505
Status	Unresolved

Description

Initially, the contract does not allow the non-excluded addresses to transfer tokens. The restriction can be resumed once the contract owner enables them.

```
if (!exemptFee[sender] && !exemptFee[recipient]) {  
    require(tradingEnabled, "Trading not enabled");  
}
```

Recommendation

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.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MSE	Missing Solidity Events	Unresolved
●	RVD	Redundant Variable Declaration	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
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	KakapoProtocol.sol#L593,598
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` and a `devWallet` as part of the transfer flow. These addresses 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.

```
uint256 marketingAmt = unitBalance * 2 * swapTaxes.marketing;
if (marketingAmt > 0) {
    payable(marketingWallet).sendValue(marketingAmt);
}

uint256 devAmt = unitBalance * 2 * swapTaxes.dev;
if (devAmt > 0) {
    payable(devWallet).sendValue(devAmt);
}
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	KakapoProtocol.sol#L689,693
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.

The contract modifies the state of some variables without checking if their current state is equal to the one given as an argument. As a result, the contract performs redundant storage writes.

```
function updateExemptFee(address _address, bool state) external onlyOwner
{
    exemptFee[_address] = state;
}

function bulkExemptFee(address[] memory accounts, bool state) external
onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        exemptFee[accounts[i]] = state;
    }
}
```

Recommendation

The team is advised to take these segments into consideration 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.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	KakapoProtocol.sol#L637,642,648,657,679,684
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function updateLiquidityProvide(bool state) external onlyOwner {
    //update liquidity providing state
    providingLiquidity = state;
}
function updateLiquidityTreshhold(uint256 new_amount) external onlyOwner {
    //update the treshhold
    require(new_amount <= 1e5, "Swap threshold amount should be lower or
    equal to % of tokens");
    tokenLiquidityThreshold = new_amount * 10**decimals();
}
...
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	KakapoProtocol.sol#L424
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.

The contract declares some variables that are not used in a meaningful way from the contract. As a result, these variables are redundant.

```
mapping(address => uint256) private _lastSell;
```

Recommendation

The team is advised to take these segments into consideration 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	KakapoProtocol.sol#L442,443
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`.

```
router  
pair
```

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	KakapoProtocol.sol#L406
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.

```
uint256 private launchtax = 99
```

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	KakapoProtocol.sol#L62,64,365,404,410,560,642,648,649,650,651,657,658,659,660,666,673,689
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 => uint256) internal _balances  
mapping(address => mapping(address => uint256)) internal _allowances  
function WETH() external pure returns (address);  
uint256 public genesis_block  
address public constant deadWallet =  
0x00000000000000000000000000000000dEaD
```

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	KakapoProtocol.sol#L424
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.

```
mapping(address => uint256) private _lastSell
```

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	KakapoProtocol.sol#L645,676
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.

```
tokenLiquidityThreshold = new_amount * 10**decimals()  
deadline = _deadline
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	KakapoProtocol.sol#L583,584,591,596
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 a loss of prediction.

```
uint256 unitBalance = deltaBalance / (denominator - swapTaxes.liquidity)
uint256 devAmt = unitBalance * 2 * swapTaxes.dev
```

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	KakapoProtocol.sol#L509,510,512
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 feeswap  
uint256 feesum  
Taxes memory currentTaxes
```

Recommendation

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

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	KakapoProtocol.sol#L9
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 ^0.8.18;
```

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	KakapoProtocol.sol#L705
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.

```
IBEP20(tokenAdd).transfer(owner(), 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

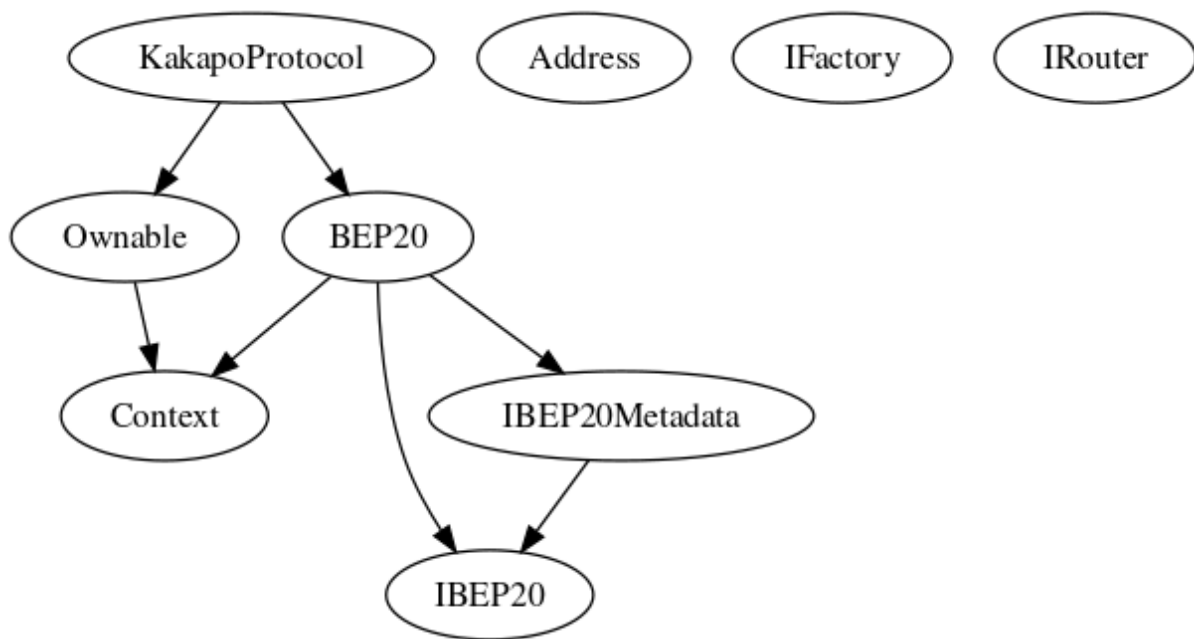
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IBEP20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IBEP20Metadata	Interface	IBEP20		
	name	External		-
	symbol	External		-
	decimals	External		-

BEP20	Implementation	Context, IBEP20, IBEP20Meta data		
		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	✓	
	_tokengeneration	Internal	✓	
	_approve	Internal	✓	
Address	Library			
	sendValue	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-

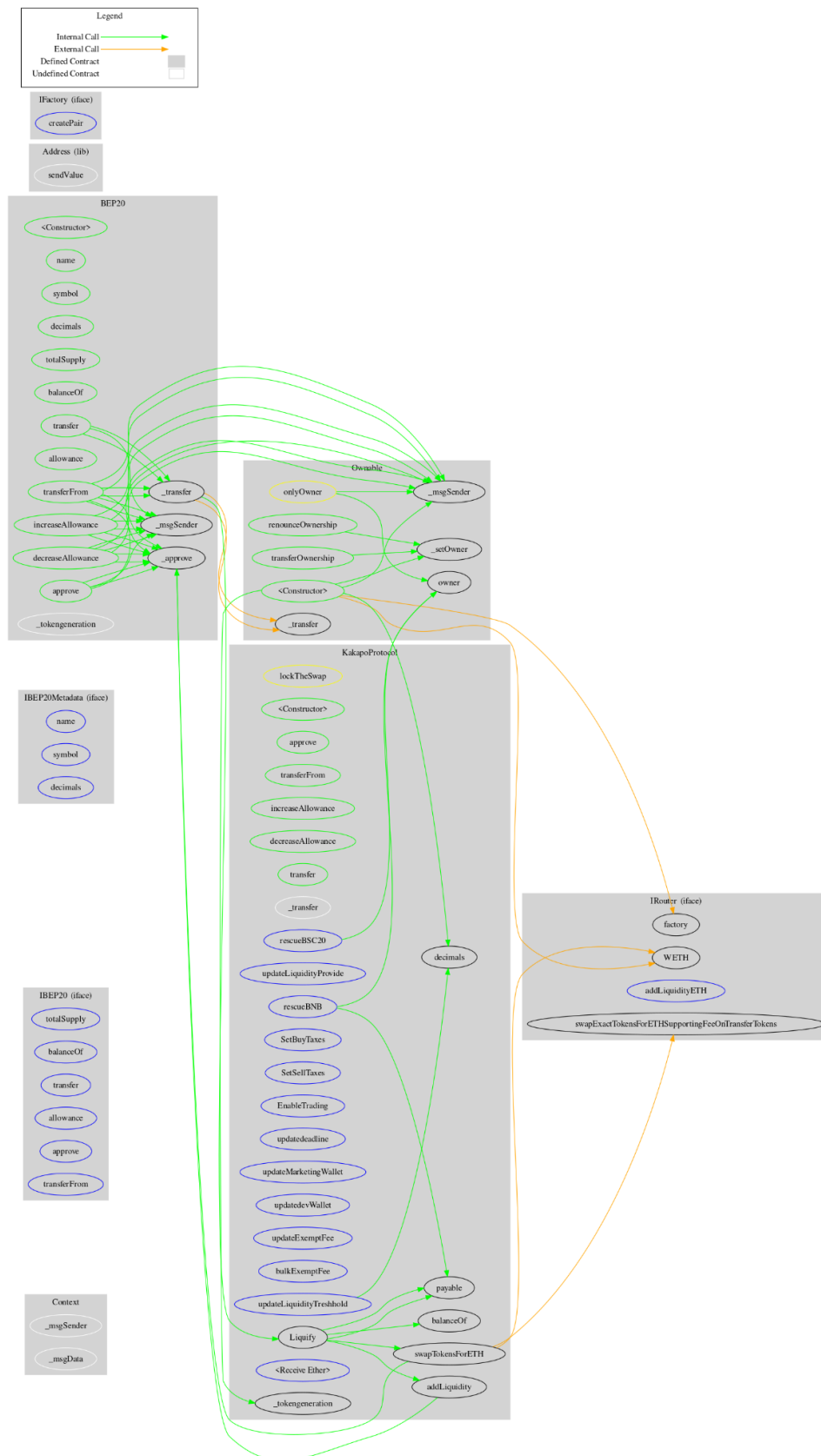
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
IFactory	Interface			
	createPair	External	✓	-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFee OnTransferTokens	External	✓	-
KakapoProtocol	Implementation	BEP20, Ownable		
		Public	✓	BEP20
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	transfer	Public	✓	-
	_transfer	Internal	✓	
	Liquify	Private	✓	lockTheSwap

	swapTokensForETH	Private	✓	
	addLiquidity	Private	✓	
	updateLiquidityProvide	External	✓	onlyOwner
	updateLiquidityTreshhold	External	✓	onlyOwner
	SetBuyTaxes	External	✓	onlyOwner
	SetSellTaxes	External	✓	onlyOwner
	EnableTrading	External	✓	onlyOwner
	updatedeadline	External	✓	onlyOwner
	updateMarketingWallet	External	✓	onlyOwner
	updatedevWallet	External	✓	onlyOwner
	updateExemptFee	External	✓	onlyOwner
	bulkExemptFee	External	✓	onlyOwner
	rescueBNB	External	✓	onlyOwner
	rescueBSC20	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

Kakapo Protocol 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. A multi-wallet signing pattern will provide security against potential hacks. There is also a limit of max 8% fees. Lastly, the contract implements a launch fee mechanism, where users are taxed with 99% fees for up to 4 blocks from launch.

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>