Cyberscope

Audit Report

Koma Inu

October 2024

Network	BSC
Address	0xd5eaAaC47bD1993d661bc087E15dfb079a7f3C19
Audited by	© cyberscope

Analysis

🔴 Critical 🟠 Medium ⚪ Minor / Informative 🔵 Pass

Severity	Code	Description	Status
🔴	ST	Stops Transactions	Unresolved
🔵	OTUT	Transfers User's Tokens	Passed
🟠	ELFM	Exceeds Fees Limit	Unresolved
🔵	MT	Mints Tokens	Passed
🔵	BT	Burns Tokens	Passed
🔵	BC	Blacklists Addresses	Passed

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	ESN	ERC20 Standard Non-Compliance	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	KOMA
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0xd5eaaac47bd1993d661bc087e15dfb079a7f3c19
Address	0xd5eaaac47bd1993d661bc087e15dfb079a7f3c19
Network	BSC
Symbol	KOMA
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

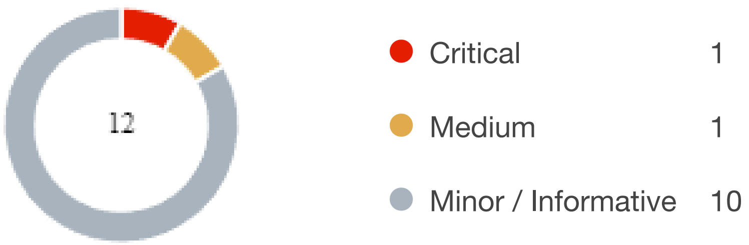
Audit Updates

Initial Audit	06 Oct 2024

Source Files

Filename	SHA256
KOMA.sol	786eca351da42edabdc744ff7f0a719b065d85cf850b4f019a0c06362e3dd8d7

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	1	0	0	0
● Medium	1	0	0	0
● Minor / Informative	10	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	KOMA.sol#L431,437
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enabled the owner will not be able to disable them again.

```
function startTrading() external onlyOwner {
        require(!tradingOpen, "Trading already enabled");
        tradingOpen = true;
        initStartTime = block.timestamp; // Set the start time
to the current block timestamp
    }

function checkTradingAllowed(
        address sender,
        address recipient
    ) internal view {
        if (!isFeeExempt[sender] && !isFeeExempt[recipient]) {
            require(tradingOpen, "ERC20: Trading is not
allowed");
        }
    }
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

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

ELFM - Exceeds Fees Limit

Criticality	Medium
Location	KOMA.sol#L268,365
Status	Unresolved

Description

During the first 30 minutes after trading is opened, the contract imposes an initial sell tax of up to 30%, which gradually decreases over time. The owner has the ability to call the setStructure function to adjust the buy and sell fees, including setting the sell fee to zero. However, even when the owner sets the sell fee to 0%, the initial sell tax mechanism remains active during the first 30 minutes, potentially resulting in sell fees exceeding the allowed 25% limit for some portion of this period.

```
function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) private {
        preTxCheck(sender, recipient, amount);
        checkTradingAllowed(sender, recipient);
        swapBack(sender, recipient, balanceOf(address(this)));
        _balances[sender] -= amount;
        uint256 amountReceived = shouldTakeFee(sender,
recipient)
            ? takeFee(sender, recipient, amount)
            : amount;
        _balances[recipient] += amountReceived;
        emit Transfer(sender, recipient, amountReceived);
    }

function getTotalFee(
        address sender,
        address recipient
    ) public view returns (uint256) {
        if (recipient == pair) {
            // Selling
            if (block.timestamp < initStartTime +
SELL_TAX_DURATION) {
                uint256 timeElapsed = block.timestamp -
initStartTime;
                uint256 sellTax = INITIAL_SELL_TAX -
                    ((INITIAL_SELL_TAX - sellFee) *
timeElapsed) /
                    SELL_TAX_DURATION;
                return sellTax;
            }
            return sellFee; // After duration, return sellFee
        }
        if (sender == pair) {
            // Buying
            if (block.timestamp < initStartTime +
BUY_TAX_DURATION) {
                uint256 timeElapsed = block.timestamp -
initStartTime;
                uint256 buyTax = INITIAL_BUY_TAX -
                    ((INITIAL_BUY_TAX - buyFee) * timeElapsed)
/
                    BUY_TAX_DURATION;
                return buyTax;
            }
            return buyFee; // After duration, return buyFee
        }
        return 0; // For transfers
    }
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

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

ESN - ERC20 Standard Non-Compliance

Criticality	Minor / Informative
Location	KOMA.sol#L251
Status	Unresolved

Description

The `preTxCheck` function in the contract includes a requirement that the transfer amount must be greater than zero. However, the ERC20 standard does not impose such a restriction, allowing transfers of zero tokens. By preventing zero-value transfers, the contract deviates from the expected behavior of an ERC20 token. Some systems and applications interacting with the contract may expect the ability to perform zero-value transfers as a legitimate operation.

```
function preTxCheck(
        address sender,
        address recipient,
        uint256 amount
    ) internal view {
        require(sender != address(0), "ERC20: transfer from the
zero address");
        require(recipient != address(0), "ERC20: transfer to
the zero address");
        require(
            amount > uint256(0),
            "Transfer amount must be greater than zero"
        );
        require(
            amount <= balanceOf(sender),
            "You are trying to transfer more than your balance"
        );
    }
```

Recommendation

It is recommended that the contract be updated to allow transfers with a zero amount to maintain full compatibility with the ERC20 standard. Removing this restriction will ensure that any ERC20-compliant systems can interact with the token without encountering issues due to this check.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	KOMA.sol#L185
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	KOMA.sol#L308
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForETH(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();
        _approve(address(this), address(router), tokenAmount);

router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	KOMA.sol#L330,463,469
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for native currency. The variable `TOKEN_SALE_THRESHOLD` 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 native currency.

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 native currency could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setTokenSaleThreshold(uint256 newThreshold) external
onlyOwner {
        require(newThreshold > 0, "Threshold must be greater
than zero");
        TOKEN_SALE_THRESHOLD = newThreshold;
        emit TokenSaleThresholdUpdated(newThreshold);
    }

function setSwapAndLiqEnabled(bool _enabled) external onlyOwner
{
        swapAndLiqEnabled = _enabled;
        emit SwapAndLiqEnabledUpdated(_enabled);
    }

function shouldSwapBack(
        address sender,
        address recipient
    ) internal view returns (bool) {
        bool aboveThreshold = balanceOf(address(this)) >=
TOKEN_SALE_THRESHOLD;
        return
            !swapping &&
            swapAndLiqEnabled &&
            !isFeeExempt[sender] &&
            recipient == pair &&
            aboveThreshold;
    }
```

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 exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	KOMA.sol#L308
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
function swapTokensForETH(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();
        _approve(address(this), address(router), tokenAmount);

router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	KOMA.sol#L145,155,156
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 _totalSupply = 1_000_000_000 * (10 **
_decimals)
uint256 private initFee = 3000
uint256 private denominator = 10000
```

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	KOMA.sol#L97,142,143,144,159,160,235,284,450,469
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);
string private constant _name = "Koma Inu"
string private constant _symbol = "KOMA"
uint8 private constant _decimals = 18
uint256 public TOKEN_SALE_THRESHOLD
address payable public marketing_receiver
address _address
bool _enabled
uint256 _sell
uint256 _buy
uint256 _amount
```

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/stable/style-guide.html#naming-conventions.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	KOMA.sol#L155
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.

```
uint256 private initFee = 3000
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	KOMA.sol#L322,326
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 max(uint256 a, uint256 b) internal pure returns
(uint256) {
        return a > b ? a : b;
    }

function min(uint256 a, uint256 b) internal pure returns
(uint256) {
        return a < b ? a : b;
    }
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	KOMA.sol#L452
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(_address).transfer(msg.sender, _amount)
```

Recommendation

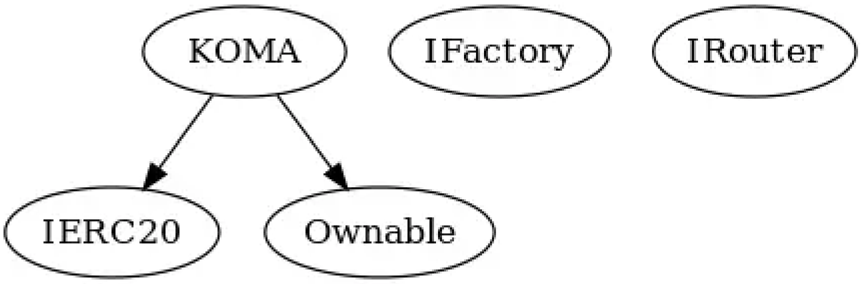
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

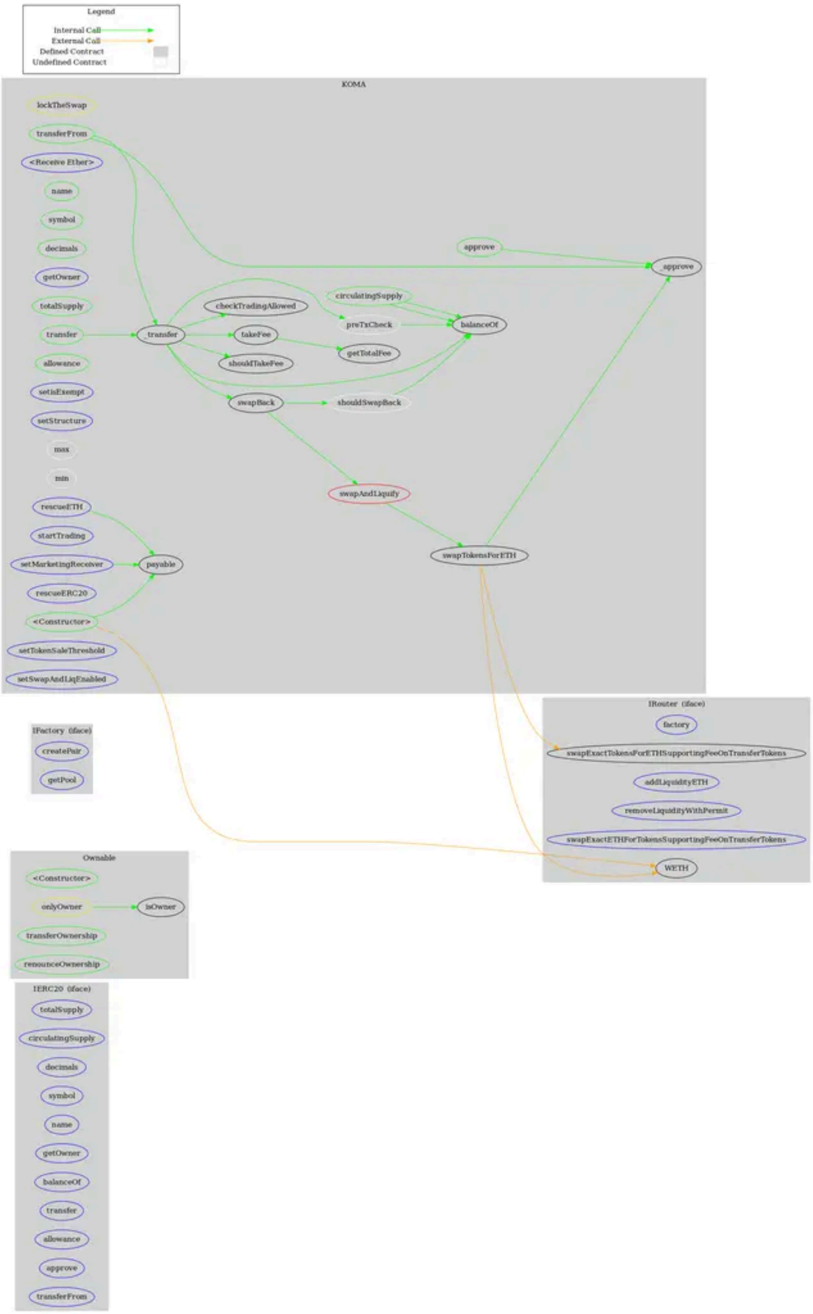
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation			
		Public	✓	-
	isOwner	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
KOMA	Implementation	IERC20, Ownable		
		Public	✓	Ownable
		External	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	getOwner	External		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	setisExempt	External	✓	onlyOwner
	approve	Public	✓	-
	circulatingSupply	Public		-

	preTxCheck	Internal		
	_transfer	Private	✓	
	setStructure	External	✓	onlyOwner
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForETH	Private	✓	
	max	Internal		
	min	Internal		
	shouldSwapBack	Internal		
	swapBack	Internal	✓	
	shouldTakeFee	Internal		
	getTotalFee	Public		-
	takeFee	Internal	✓	
	transferFrom	Public	✓	-
	_approve	Private	✓	
	startTrading	External	✓	onlyOwner
	checkTradingAllowed	Internal		
	rescueETH	External	✓	onlyOwner
	rescueERC20	External	✓	onlyOwner
	setMarketingReceiver	External	✓	onlyOwner
	setTokenSaleThreshold	External	✓	onlyOwner
	setSwapAndLiqEnabled	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
KOMA
lockTheSwap
transferFrom
<Receive Ether>
name
symbol
decimals
getOwner
totalSupply
transfer
allowance
setisExempt
setStructure
max
min
rescueETH
startTrading
setMarketingReceiver
rescueERC20
<Constructor>
setTokenSaleThreshold
setSwapAndLiqEnabled
approve
_approve
checkTradingAllowed
circulatingSupply
preTxCheck
balanceOf
_transfer
takeFee
getTotalFee
shouldTakeFee
swapBack
shouldSwapBack
swapAndLiquify
payable
swapTokensForETH
IFactory (iface)
createPair
getPool
IRouter (iface)
factory
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidityETH
removeLiquidityWithPermit
swapExactETHForTokensSupportingFeeOnTransferTokens
WETH
Ownable
<Constructor>
onlyOwner
isOwner
transferOwnership
renounceOwnership
IERC20 (iface)
totalSupply
circulatingSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom

Summary

Koma Inu 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 stop transactions and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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

cyberscope.io