Audit Report

Meme Casino

October 2023

SHA256 f61e28775775fa8b319433a416d1680c199fbb00dae4e354a925fc27a36ec84b

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	US	Untrusted Source	Unresolved
●	IRC	Inconsistent Ratio Calculation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MC	Missing Check	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	MemeCasino
Testing Deploy	https://testnet.bscscan.com/address/0xd2b412dd393c17a32c8306be3f5cce469c107dc3
Symbol	MEMEC
Decimals	18
Total Supply	1,000,000

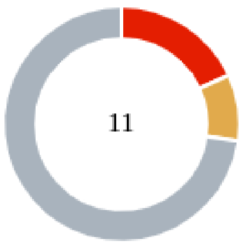
Audit Updates

Initial Audit	27 Oct 2023

Source Files

Filename	SHA256
MemeCasino.sol	f61e28775775fa8b319433a416d1680c199fbb00dae4e354a925fc27a36ec84b

Findings Breakdown



●	Critical	2
●	Medium	1
●	Minor / Informative	8

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	8	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	MemeCasino.sol#L483
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 enable the owner will not be able to disable them again.

```
if(!tradingEnabled) {
    if (!other) {
        revert("Trading not yet enabled!");
    } else if (!_isExcludedFromProtection[from] &&
!_isExcludedFromProtectio[to]) {
        revert("Tokens cannot be moved until trading is
live.");
    }
}
```

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.

US - Untrusted Source

Criticality	Critical
Location	MemeCasino.sol#L349,388,571,704
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions. Specifically, the contract uses the external `initializer` for critical functionalities and assignments of key variables.

```
   function setInitializer(address init) public onlyOwner {
        require(!tradingEnabled);
        require(init != address(this), "Can't be self.");
        initializer = Initializer(init);
        try initializer.getConfig() returns (address router,
address constructorLP) {
            dexRouter = IRouter02(router); lpPair =
constructorLP; lpPairs[lpPair] = true;
            _approve(_owner, address(dexRouter),
type(uint256).max);
            _approve(address(this), address(dexRouter),
type(uint256).max);
        } catch { revert(); }
    }

    function removeSniper(address account) external onlyOwner {
        initializer.removeSniper(account);
    }

    function setProtectionSettings(bool _antiSnipe, bool
_antiBlock) external onlyOwner {
        initializer.setProtections(_antiSnipe, _antiBlock);
    }

    ...
    try initializer.setLaunch(lpPair, uint32(block.number),
uint64(block.timestamp), _decimals) {} catch {}
        try initializer.getInits(balanceOf(lpPair)) returns
(uint256 initThreshold, uint256 initSwapAmount) {
            swapThreshold = initThreshold;
            swapAmount = initSwapAmount;
        } catch {}

    ....
    if (_hasLimits(from, to)) { bool checked;
        try initializer.checkUser(from, to, tAmount) returns
(bool check) {
            checked = check; } catch { revert(); }
        if(!checked) { revert(); }
    ...
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

IRC - Inconsistent Ratio Calculation

Criticality	Medium
Location	MemeCasino.sol#L146,700
Status	Unresolved

Description

The contract is designed to initialize various ratios, which, when summed up, equal the `totalSwap` variable. These ratios include values for `reflection`, `development`, `jackpot`, `burn`, and `marketing`. However, there's an inconsistency in the calculation of the `total` variable. Instead of just considering the `totalSwap` value, the contract erroneously adds the `ratios.reflection` and `ratios.burn values` to it. This leads to a discrepancy between the expected total (as represented by `totalSwap`) and the calculated total.

```
    Ratios public _ratios = Ratios({
            reflection: 50,
            development: 150,
            jackpot: 200,
            burn: 50,
            marketing: 50,
            totalSwap: 500
        });

        ...
        uint256 total = ratios.totalSwap + ratios.reflection + ratios.burn;
        ...
```

Recommendation

It is recommended to address this inconsistency to ensure accurate ratio calculations. Two potential solutions are:

1. Adjust the `totalSwap` value to `400`, so that when the `ratios.reflection` and `ratios.burn` values are added, the total becomes `500`.

2. If the intention is for `totalSwap` to represent the complete total of `500`, then the calculation for the `total` variable should only consider `ratios.totalSwap`, without adding the `ratios.reflection` and `ratios.burn` values.

Properly aligning these values ensures clarity and prevents potential issues in contract operations.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	MemeCasino.sol#L364
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
   function setExcludedFromLimits(address account, bool
enabled) external onlyOwner {
        _isExcludedFromLimits[account] = enabled;
    }

    function setExcludedFromFees(address account, bool enabled)
public onlyOwner {
        _isExcludedFromFees[account] = enabled;
    }

    function setExcludedFromProtection(address account, bool
enabled) external onlyOwner {
        _isExcludedFromProtection[account] = enabled;
    }

    function removeSniper(address account) external onlyOwner {
        initializer.removeSniper(account);
    }

    function setProtectionSettings(bool _antiSnipe, bool
_antiBlock) external onlyOwner {
        initializer.setProtections(_antiSnipe, _antiBlock);
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

MC - Missing Check

Criticality	Minor / Informative
Location	MemeCasino.sol#L412
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically the parameters `thresholdDivisor` and `amountDivisor` can be set to zero.

```
 function setSwapSettings(uint256 thresholdPercent, uint256
thresholdDivisor, uint256 amountPercent, uint256 amountDivisor)
external onlyOwner {
        swapThreshold = (_tTotal * thresholdPercent) /
thresholdDivisor;
        swapAmount = (_tTotal * amountPercent) / amountDivisor;
        ...
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MemeCasino.sol#L33,117,118,119,120,121,140,146,155,167,180,392
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 constant private startingSupply = 1_000_000
string constant private _name = "Meme Casino"
string constant private _symbol = "MEMEC"
uint8 constant private _decimals = 18
uint256 constant private _tTotal = startingSupply *
10**_decimals

Fees public _taxRates = Fees({
        buyFee: 500,
        sellFee: 500,
        transferFee: 0
    })

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	MemeCasino.sol#L413,423
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.

```
swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor
piSwapPercent = priceImpactSwapPercent
```

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	MemeCasino.sol#L722,723,724
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 feeAmount = (tAmount * currentFee) / masterTaxDivisor
values.tFee = (feeAmount * ratios.reflection) / total
```

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	MemeCasino.sol#L353,572,694,705,706
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.

```
address router
address constructorLP
uint256 initSwapAmount
uint256 initThreshold
ExtraValues memory values
bool checked
bool check
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MemeCasino.sol#L259
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.

```
operator = newOperator
```

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	MemeCasino.sol#L591
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.

```
TOKEN.transfer(_owner, TOKEN.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**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IFactoryV2	Interface			
	getPair	External		-
	createPair	External	✓	-
IV2Pair	Interface			
	factory	External		-

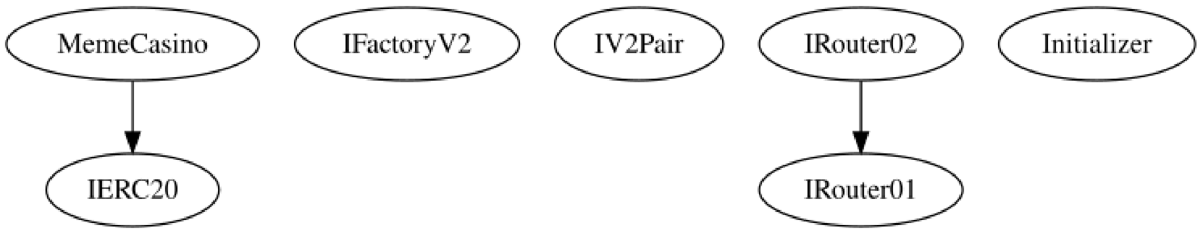
	getReserves	External		-
	sync	External	✓	-
IRouter01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	addLiquidity	External	✓	-
	swapExactETHForTokens	External	Payable	-
	getAmountsOut	External		-
	getAmountsIn	External		-
IRouter02	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
Initializer	Interface			
	setLaunch	External	✓	-
	getConfig	External	✓	-
	getInits	External	✓	-
	setLpPair	External	✓	-

	checkUser	External	✓	-
	setProtections	External	✓	-
	removeSniper	External	✓	-
MemeCasino	Implementation	IERC20		
		Public	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
	setOperator	Public	✓	-
	renounceOriginalDeployer	External	✓	-
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	External	✓	onlyOwner
	transferFrom	External	✓	-

	setNewRouter	External	✓	onlyOwner
	setLpPair	External	✓	onlyOwner
	setInitializer	Public	✓	onlyOwner
	isExcludedFromLimits	External		-
	setExcludedFromLimits	External	✓	onlyOwner
	isExcludedFromFees	External		-
	setExcludedFromFees	Public	✓	onlyOwner
	isExcludedFromProtection	External		-
	setExcludedFromProtection	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	removeSniper	External	✓	onlyOwner
	setProtectionSettings	External	✓	onlyOwner
	lockTaxes	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	getTokenAmountAtPriceImpact	External		-
	setSwapSettings	External	✓	onlyOwner
	setPriceImpactSwapAmount	External	✓	onlyOwner
	setContractSwapEnabled	External	✓	onlyOwner
	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_transfer	Internal	✓	
	contractSwap	Internal	✓	inSwapFlag
	_checkLiquidityAdd	Internal	✓	

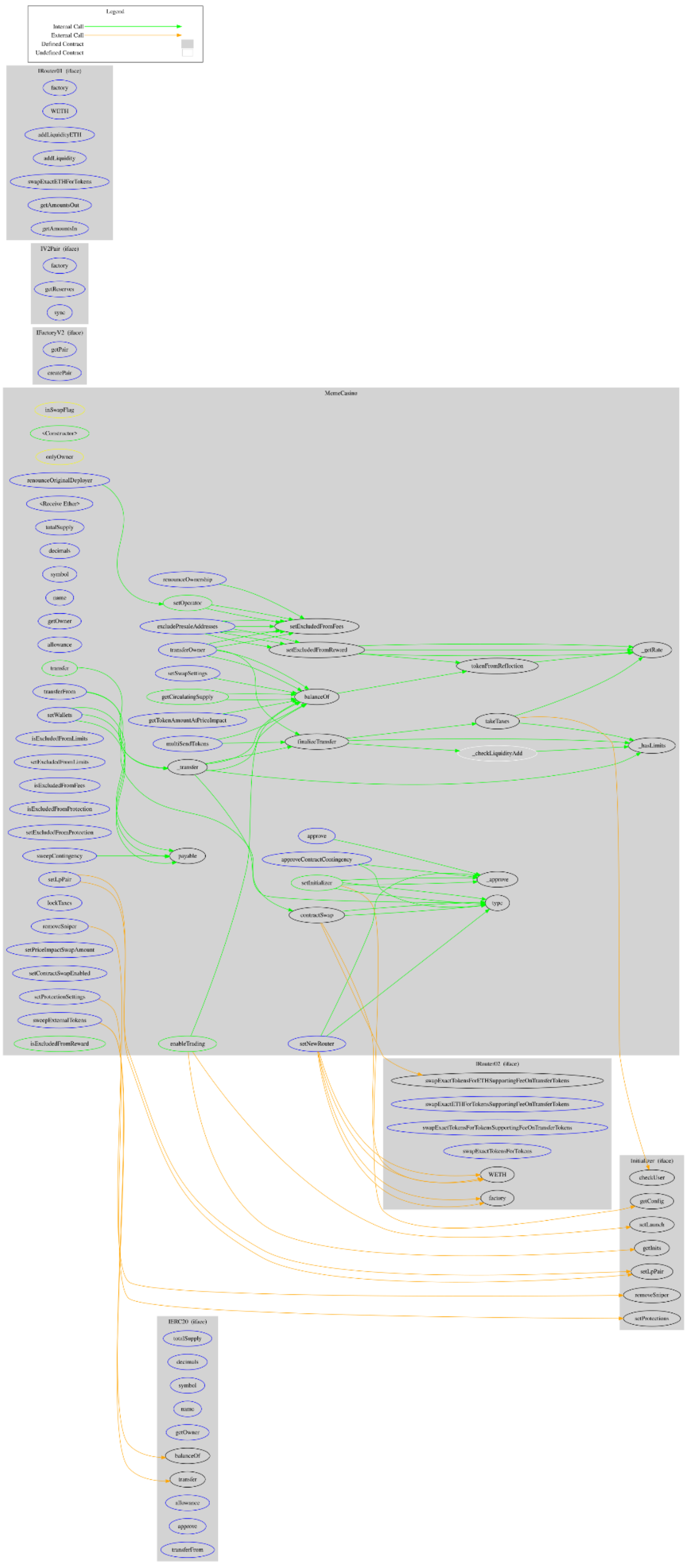
	enableTrading	Public	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	sweepExternalTokens	External	✓	onlyOwner
	multiSendTokens	External	✓	onlyOwner
	isExcludedFromReward	Public		-
	setExcludedFromReward	Public	✓	onlyOwner
	tokenFromReflection	Public		-
	finalizeTransfer	Internal	✓	
	takeTaxes	Internal	✓	
	_getRate	Internal		

Inheritance Graph



$$\begin{tikzcd}
\text{MemeCasino} \arrow[d] & \text{IFactoryV2} & \text{IV2Pair} & \text{IRouter02} \arrow[d] & \text{Initializer} \\
\text{IERC20} & & & \text{IRouter01} &
\end{tikzcd}$$

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
swapExactETHForTokens
getAmountsOut
getAmountsIn
IV2Pair (iface)
factory
getReserves
sync
IFactoryV2 (iface)
getPair
createPair
MemeCasino
inSwapFlag
<Constructor>
onlyOwner
renounceOriginalDeployer
<Receive Ether>
totalSupply
decimals
symbol
name
getOwner
allowance
transfer
transferFrom
setWallets
isExcludedFromLimits
setExcludedFromLimits
isExcludedFromFees
isExcludedFromProtection
setExcludedFromProtection
sweepContingency
setLpPair
lockTaxes
removeSniper
setPriceImpactSwapAmount
setContractSwapEnabled
setProtectionSettings
sweepExternalTokens
isExcludedFromReward
renounceOwnership
setOperator
excludePresaleAddresses
transferOwner
setSwapSettings
getCirculatingSupply
getTokenAmountAtPriceImpact
multiSendTokens
_transfer
payable
enableTrading
setExcludedFromFees
setExcludedFromReward
balanceOf
finalizeTransfer
approve
approveContractContingency
setInitializer
contractSwap
setNewRouter
tokenFromReflection
takeTaxes
_checkLiquidityAdd
_approve
type
_getRate
_hasLimits
IRouter02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens
WETH
factory
Initializer (iface)
checkUser
getConfig
setLaunch
getInits
setLpPair
removeSniper
setProtections
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom

Summary

Meme Casino contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The contract uses an external initializer for critical functionalities and assignments of key variables. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 5% fees on buy and sell transactions.

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