Cyberscope

# Audit Report

# R33LZ

January 2023

| | |
|---|---|
| Type | ERC20 |
| Network | ETH |
| Address | 0x5D3957DF518adE9AdfD7c4341a0cf7D6D73EC682 |
| Audited by | © cyberscope |

# Table of Contents

# Review

| | |
|---|---|
| **Contract Name** | R33lz |
| **Compiler Version** | v0.8.17+commit.8df45f5f |
| **Optimization** | 200 runs |
| **Explorer** | https://etherscan.io/address/0x5d3957df518ade9adfd7c4341a0cf7d6d73ec682 |
| **Address** | 0x5d3957df518ade9adfd7c4341a0cf7d6d73ec682 |
| **Network** | ETH |
| **Symbol** | R33LZ |
| **Decimals** | 18 |
| **Total Supply** | 1,000,000,000 |

# Audit Updates

| | |
|---|---|
| **Initial Audit** | 19 Jan 2023 |

# Source Files

| **Filename** | SHA256 |
|---|---|
| **R33lz.sol** | b312ff0d6b0218c2e13d6c9bab0e75e5b831b86103b9b54e51f0ffd1b224f83b |

# Analysis

● Critical ● Medium ● Minor / Informative ● Pass

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | ST | Stops Transactions | Passed |
| ● | 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 |

# Diagnostics

Critical Medium Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | PTRP | Potential Transfer Revert Propagation | Unresolved |
| ● | DDP | Decimal Division Precision | Unresolved |
| ● | MC | Missing Check | Unresolved |
| ● | CO | Code Optimization | 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 |
| ● | L19 | Stable Compiler Version | Unresolved |
| ● | L20 | Succeeded Transfer Check | Unresolved |

# PTRP - Potential Transfer Revert Propagation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | R33lz.sol#L620,623 |
| **Status** | Unresolved |

## Description

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

```
if (marketingAmt > 0) {
    payable(marketingWallet).sendValue(marketingAmt);
}
if (developerAmt > 0) {
    payable(devWallet).sendValue(developerAmt);
}
```

## 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 archived by not allowing set contract addresses or by sending the funds in a non-revertable way.

# DDP - Decimal Division Precision

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | R33lz.sol#L587 |
| **Status** | Unresolved |

## Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 unitBalance = deltaBalance / (denominator -
swapTaxes.liquidity);
uint256 ethToAddLiquidityWith = unitBalance * swapTaxes.liquidity;

if (ethToAddLiquidityWith > 0) {
    // Add liquidity to pancake
    addLiquidity(tokensToAddLiquidityWith, ethToAddLiquidityWith);
}

uint256 marketingAmt = unitBalance * 2 * swapTaxes.marketing;
uint256 developerAmt = unitBalance * 2 * swapTaxes.developer;
```

## Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

# MC - Missing Check

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | R33lz.sol#L701 |
| **Status** | Unresolved |

## Description

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

```
function updateRouterAndPair(address newRouter, address newPair)
external onlyOwner {
    router = IRouter(newRouter);
    pair = newPair;
}
```

## Recommendation

The team is advised to properly check the variables according to the required specifications. The address arguments should not be set to zero address.

# CO - Code Optimization

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | R33lz.sol#L536 |
| **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 variables `feeswap` and `feesum` hold the same information. Hence, one variable is unnecessary and redundant.

```
uint256 feeswap;
uint256 feesum;

feeswap =
    sellTaxes.liquidity +
    sellTaxes.marketing +
    sellTaxes.developer;
    feesum = feeswap;
`
```

## Recommendation

The team is advised to take into consideration these segments 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.

# L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | R33lz.sol#L62,64,365,411,583,664,670,671,672,673,679,680,681,682,688,689,690,691,702,709,716,726 |
| **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);
address public constant deadWallet =
0x000000000000000000000000000000000000dEaD

...
```

## 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** | R33lz.sol#L667,736 |
| **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()
maxBuyLimit = maxBuy * 10**decimals()
```

## 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** | R33lz.sol#L606,607,614,615 |
| **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 unitBalance = deltaBalance / (denominator -
swapTaxes.liquidity)
uint256 ethToAddLiquidityWith = unitBalance * swapTaxes.liquidity
```

## 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** | R33lz.sol#L532,533,535 |
| **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.

# L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | R33lz.sol#L699 |
| **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.

```
pair = newPair
```

## 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.

# L19 - Stable Compiler Version

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | R33lz.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 ^0.8.17;
```

## 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** | R33lz.sol#L747 |
| **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(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.

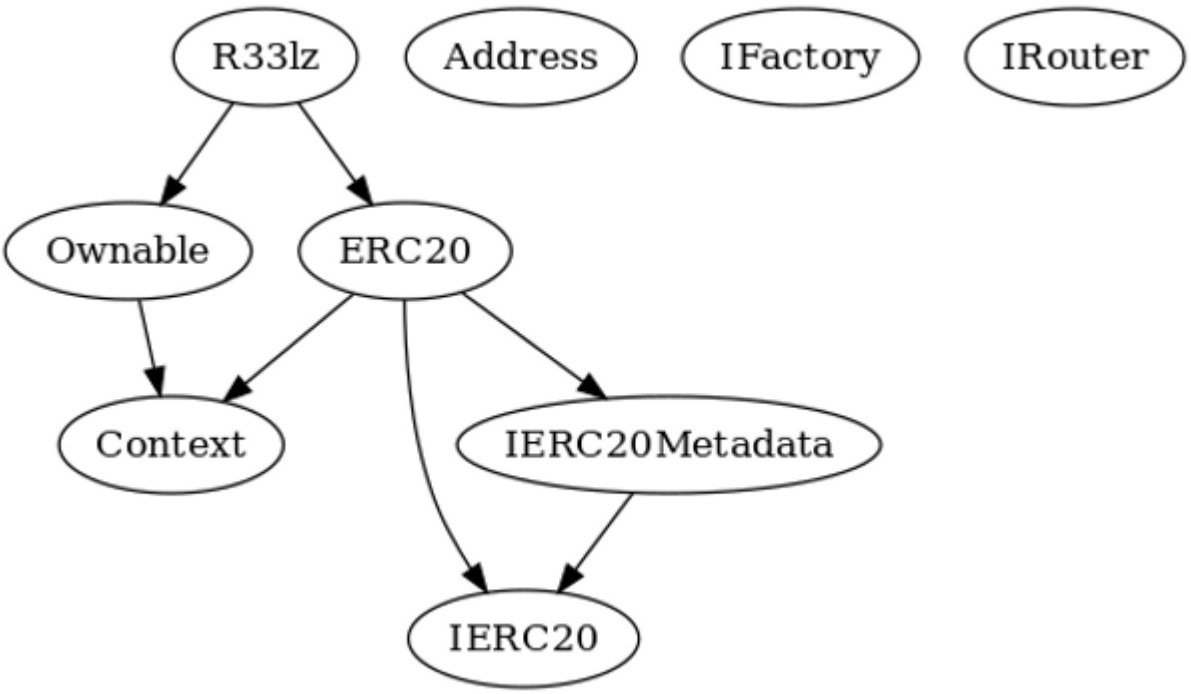
# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **Context** | Implementation | | | |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **IERC20Metadata** | Interface | IERC20 | | |
| | name | External | | - |
| | symbol | External | | - |
| | decimals | External | | - |
| | | | | |
| **ERC20** | Implementation | Context, IERC20, IERC20Metadata | | |
| | | Public | ✓ | - |
| | name | Public | | - |
| | symbol | Public | | - |
| | decimals | Public | | - |

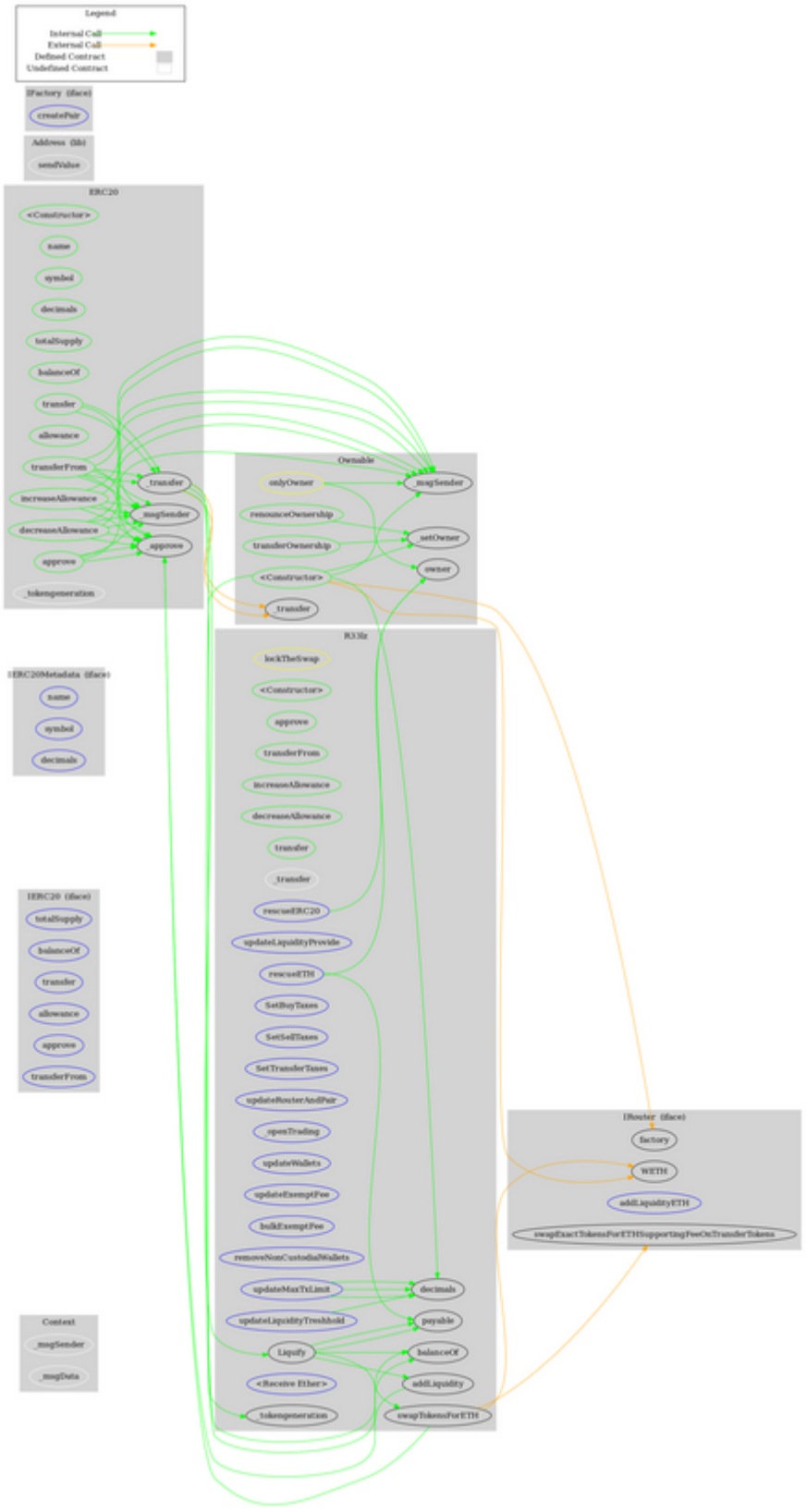
| | | | | |
|---|---|---|---|---|
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | allowance | Public | | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | _transfer | Internal | ✓ | |
| | _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 | - |

| | | | | |
|---|---|---|---|---|
| | swapExactTokensForETHSupporting FeeOnTransferTokens | External | ✓ | - |
| | | | | |
| **R33lz** | Implementation | ERC20, Ownable | | |
| | | Public | ✓ | ERC20 |
| | 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 |
| | SetTransferTaxes | External | ✓ | onlyOwner |
| | updateRouterAndPair | External | ✓ | onlyOwner |
| | _openTrading | External | ✓ | onlyOwner |
| | updateWallets | External | ✓ | onlyOwner |
| | updateExemptFee | External | ✓ | onlyOwner |
| | bulkExemptFee | External | ✓ | onlyOwner |
| | removeNonCustodialWallets | External | ✓ | onlyOwner |
| | updateMaxTxLimit | External | ✓ | onlyOwner |
| | rescueETH | External | ✓ | onlyOwner |
| | rescueERC20 | External | ✓ | onlyOwner |
| | | External | Payable | - |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IFactory (iface)
createPair
Address (lib)
sendValue
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_tokengeneration
_transfer
_msgSender
_approve
Ownable
onlyOwner
renounceOwnership
transferOwnership
<Constructor>
transfer
_msgSender
setOwner
owner
R33lz
lockTheSwap
<Constructor>
approve
transferFrom
increaseAllowance
decreaseAllowance
transfer
_transfer
rescueERC20
updateLiquidityProvide
rescueETH
SetBuyTaxes
SetSellTaxes
SetTransferTaxes
updateRouterAndPair
openTrading
updateWallets
updateExemptFee
bulkExemptFee
removeNonCustodialWallets
updateMaxTxLimit
updateLiquidityTreshhold
Liquify
<Receive Ether>
_tokengeneration
decimals
payable
balanceOf
addLiquidity
swapTokensForETH
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
IRouter (iface)
factory
WETH
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens

# Summary

R33LZ is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. The contract has a 33% tax on sales until the owner disables it. There is also a limit of max 15% fees.

# 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