



Cyberscope

Audit Report

PEPE

September 2023

Network ETH

Address 0x70040E97CE20581Ed121BE077FCc6D7AAFe39C34

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	CR	Code Repetition	Unresolved
●	RVA	Redundant Variable Assignment	Unresolved
●	RSML	Redundant SafeMath Library	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
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	5
Audit Updates	5
Source Files	5
Findings Breakdown	6
CR - Code Repetition	7
Description	7
Recommendation	9
RVA - Redundant Variable Assignment	11
Description	11
Recommendation	11
RSM - Redundant SafeMath Library	12
Description	12
Recommendation	12
L02 - State Variables could be Declared Constant	13
Description	13
Recommendation	13
L04 - Conformance to Solidity Naming Conventions	14
Description	14
Recommendation	15
L05 - Unused State Variable	16
Description	16
Recommendation	16
L07 - Missing Events Arithmetic	17
Description	17
Recommendation	18
L09 - Dead Code Elimination	19
Description	19
Recommendation	20
L13 - Divide before Multiply Operation	21
Description	21
Recommendation	21
L16 - Validate Variable Setters	22
Description	22
Recommendation	22
L19 - Stable Compiler Version	23
Description	23

Recommendation	23
Functions Analysis	24
Inheritance Graph	32
Flow Graph	33
Summary	34
Disclaimer	35
About Cyberscope	36

Review

Contract Name	PEPE
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://etherscan.io/address/0x70040e97ce20581ed121be077fcc6d7aafe39c34
Address	0x70040e97ce20581ed121be077fcc6d7aafe39c34
Network	ETH
Symbol	PEPE
Decimals	18
Total Supply	420,690,000,000,000

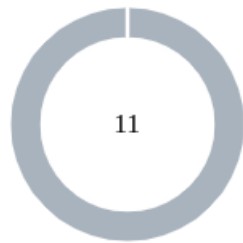
Audit Updates

Initial Audit	13 Sep 2023
---------------	-------------

Source Files

Filename	SHA256
PEPE.sol	4c9f4fc7a725e3506598910a0b838a06c149670a5910471f2bd904c8b878f058

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	11	0	0	0

CR - Code Repetition

Criticality	Minor / Informative
Location	PEPE.sol#L1442,1480
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.

Specifically the `setSellTax` and `setBuyTax` functions, as well as the `setMaxTx` and `setMaxWallet` functions share identical code segments.


```
function setSellTax(
    uint256 _wholeNumber,
    uint256 _firstNumberAfterDecimal,
    uint256 _secondNumberAfterDecimal
) public onlyOwner {
    require(
        _wholeNumber < 100 &&
        _firstNumberAfterDecimal <= 9 &&
        _secondNumberAfterDecimal <= 9
    );
    centiSellTax =
        _wholeNumber *
        100 +
        _firstNumberAfterDecimal *
        10 +
        _secondNumberAfterDecimal;
}

function setBuyTax(
    uint256 _wholeNumber,
    uint256 _firstNumberAfterDecimal,
    uint256 _secondNumberAfterDecimal
) public onlyOwner {
    require(
        _wholeNumber < 100 &&
        _firstNumberAfterDecimal <= 9 &&
        _secondNumberAfterDecimal <= 9
    );
    centiBuyTax =
        _wholeNumber *
        100 +
        _firstNumberAfterDecimal *
        10 +
        _secondNumberAfterDecimal;
}
```

```
function setMaxTx(  
    uint256 _wholeNumber,  
    uint256 _firstNumberAfterDecimal,  
    uint256 _secondNumberAfterDecimal  
) external onlyOwner {  
    require(  
        _wholeNumber < 100 &&  
        _firstNumberAfterDecimal <= 9 &&  
        _secondNumberAfterDecimal <= 9  
    );  
    maxTxAmount =  
        (_wholeNumber *  
            100 +  
            _firstNumberAfterDecimal *  
            10 +  
            _secondNumberAfterDecimal) *  
            totalSupply().div(10000);  
}  
  
function setMaxWallet(  
    uint256 _wholeNumber,  
    uint256 _firstNumberAfterDecimal,  
    uint256 _secondNumberAfterDecimal  
) external onlyOwner {  
    require(  
        _wholeNumber < 100 &&  
        _firstNumberAfterDecimal <= 9 &&  
        _secondNumberAfterDecimal <= 9  
    );  
    maxWalletAmount =  
        (_wholeNumber *  
            100 +  
            _firstNumberAfterDecimal *  
            10 +  
            _secondNumberAfterDecimal) *  
            totalSupply().div(10000);  
}
```

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.

RVA - Redundant Variable Assignment

Criticality	Minor / Informative
Location	PEPE.sol#L1180
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.

Specifically, the contract declares the `Optimization` variable, initialized with the value `169450327845919094`. However this variable is not utilized in any of the contract's functions or logic. As a result, the `Optimization` variable serves no functional purpose within the contract and is redundant.

```
uint256 public Optimization = 169450327845919094
```

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.

The contract should remove the `Optimization` variable from the contract to reduce complexity and save on gas costs.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	PEPE.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 to 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 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>.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	PEPE.sol#L1180
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 public Optimization = 169450327845919094
```

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	PEPE.sol#L705,707,738,780,1180,1443,1444,1445,1462,1463,1464,1481,1482,1483,1501,1502,1503
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 PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
function WETH() external pure returns (address);
uint256 public Optimization = 169450327845919094
uint256 _wholeNumber
uint256 _firstNumberAfterDecimal
uint256 _secondNumberAfterDecimal
```

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	PEPE.sol#L1082
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.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

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	PEPE.sol#L1439,1452,1471,1490,1510
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.

```
swapTokensAtAmount = amount

centiSellTax =
    _wholeNumber *
    100 +
    _firstNumberAfterDecimal *
    10 +
    _secondNumberAfterDecimal

centiBuyTax =
    _wholeNumber *
    100 +
    _firstNumberAfterDecimal *
    10 +
    _secondNumberAfterDecimal

...
```

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	PEPE.sol#L582,980,988,1128,1133,1144
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 _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    _balances[account] = _balances[account].sub(
        amount,
        "ERC20: burn amount exceeds balance"
    );
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	PEPE.sol#L1254,1258,1490,1510
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 precision.

```
maxTxAmount =  
    parameters.maxTxPercent *  
    supply_ *  
    (10 ** decimals_) .div(10000)
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PEPE.sol#L1006,1232,1281,1302
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.

```
_owner = msgSender
payable(addr_).transfer(msg.value)
uniswapV2Pair = _uniswapV2Pair
marketingWallet = wallet
```

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	PEPE.sol#L8
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 ^0.8.15;
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
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, IERC20Meta 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	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-

	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			

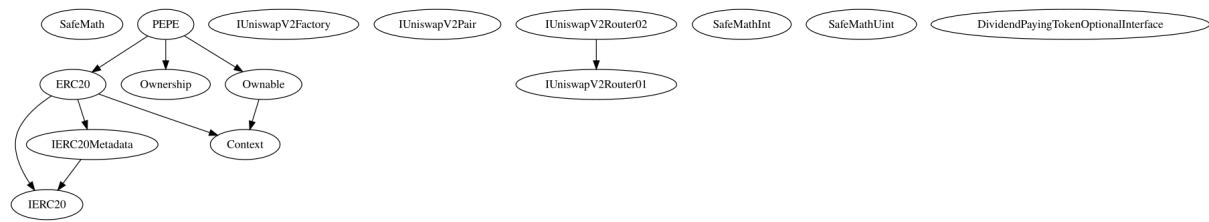
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-

	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Ownership	Implementation			
		Public	✓	-
	addr	Internal		
	fee	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		

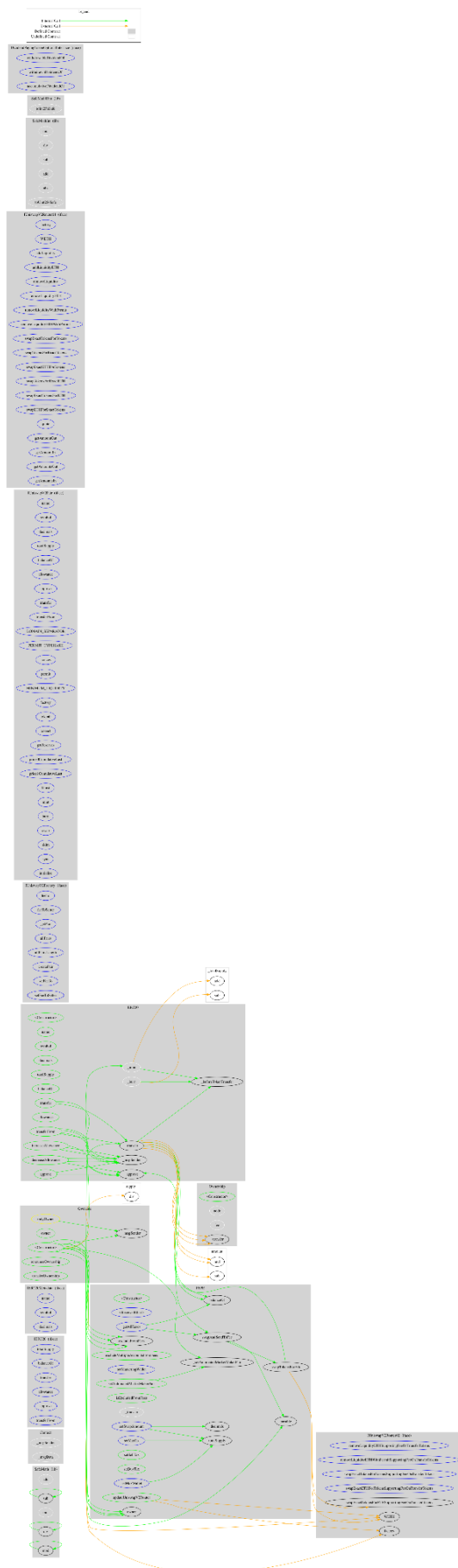
SafeMathUint	Library			
	toInt256Safe	Internal		
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
PEPE	Implementation	ERC20, Ownable, Ownership		
		Public	Payable	ERC20 Ownership
		External	Payable	-
	updateUniswapV2Router	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapAndSendToFee	Private	✓	
	swapTokensForEth	Private	✓	
	getAllTaxes	External	✓	onlyOwner

	setSwapAmount	External	✓	onlyOwner
	setSellTax	Public	✓	onlyOwner
	setBuyTax	Public	✓	onlyOwner
	setMaxTx	External	✓	onlyOwner
	setMaxWallet	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

PEPE contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. PEPE is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

<https://etherscan.io/tx/0x90493094b4ca1c30e44dc95f09f86b3e2aff812b965fe7dd2d9c81cbde9fc84>

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>