Cyberscope

Audit Report

$DRIP

Aug 2023

SHA256 dfe38b70103dcc4c09a74987a7feae695f467b857dae07f59f4cdd79e0d40a68

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
🔴	USF	Unlocked Swap Functionality	Unresolved
🔴	US	Untrusted Source	Unresolved
⚪	RAV	Router Argument Validation	Unresolved
⚪	OCTD	Transfers Contract's Tokens	Unresolved
⚪	MEE	Missing Events Emission	Unresolved
⚪	RSW	Redundant Storage Writes	Unresolved
⚪	MVN	Misleading Variables Naming	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L13	Divide before Multiply Operation	Unresolved
⚪	L14	Uninitialized Variables in Local Scope	Unresolved

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x24dd76ac0976adee24be1e0160795622be9ed92a

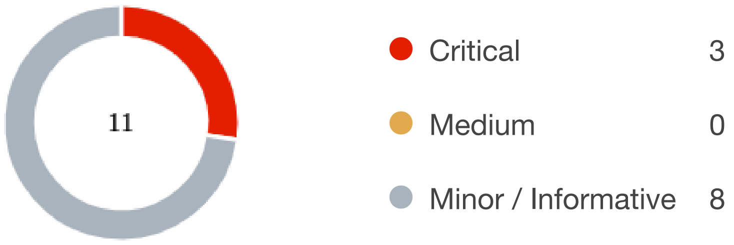
Audit Updates

Initial Audit	02 Aug 2023 https://github.com/cyberscope-io/audits/blob/main/1-drip/v1/audit.pdf
Corrected Phase 2	06 Aug 2023

Source Files

Filename	SHA256
contracts/DripToken.sol	dfe38b70103dcc4c09a74987a7feae695f467b857dae07f59f4cdd79e0d40a68

Findings Breakdown



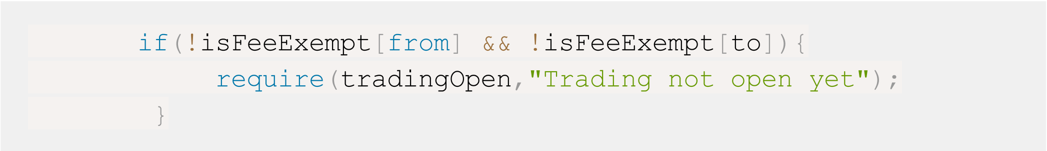
Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	3	0	0	0
● Medium	0	0	0	0
● Minor / Informative	8	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	DripToken.sol#L335
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(!isFeeExempt[from] && !isFeeExempt[to]){
            require(tradingOpen,"Trading not open yet");
        }
```

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.

USF - Unlocked Swap Functionality

Criticality	Critical
Location	contracts/DripToken.sol#L316
Status	Unresolved

Description

The smart contract contains the `transferTaxes` function which attempts to execute the `performTaxSwap` function within the `taxContract`.

The contract is designed with the intention of performing a swap operation, as suggested by the name of the `performTaxSwap` function. Despite this, the swap operation is not locked during the transaction. This is a significant discrepancy between the contract's design and its actual implementation.

The absence of a locking mechanism during the swap operation can lead to potential infinite loop.

```
   function transferTaxes() internal {
        uint256 amountToTransfer = balanceOf(address(this));
        _transfer(address(this), address(taxContract),
amountToTransfer);
        try taxContract.performTaxSwap() {
        } catch {
        }
    }
```

Recommendation

It is recommended to lock the swap operation during the transaction. This can be achieved by introducing a modifier that locks the swap operation when the `performTaxSwap` function is called and unlocks it once the operation is complete. This would ensure that the swap operation cannot be called again until the first swap operation has finished, preventing potential infinite loops.

US - Untrusted Source

Criticality	Critical
Location	DripToken.sol#L207
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.

```
    function updateTaxContract (ITaxContract _taxContract) 
external onlyOwner {
        taxContract = ITaxContract(_taxContract);
    }
```

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.

RAV - Router Argument Validation

Criticality	Minor / Informative
Location	DripToken.sol#L200
Status	Unresolved

Description

The contract does not validate the `_router` address that is passed as parameter to the `updateRouter` function. This lack of validation can lead to unintended behavior and potential security vulnerabilities.

```
    function updateRouter(address _router) external onlyOwner {
        uniswapV2Router = IUniswapV2Router(_router);
    }
```

Recommendation

It is recommended to add validation checks for the router address. These checks should include verifying that the address is not null, and that the pair address associated with the router has a valid pair with the router's native token.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	DripToken.sol#L229
Status	Unresolved

Description

The contract owner has the authority to claim all the token balance of the contract. The owner may take advantage of it by calling the `clearStuckToken` function.

```
   function clearStuckToken(address tokenAddress, uint256 
tokens) external onlyOwner returns (bool success) {
        if(tokens == 0){
            tokens = 
IERC20(tokenAddress).balanceOf(address(this));
        }
        return IERC20(tokenAddress).transfer(msg.sender, 
tokens);
    }
```

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 time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	DripToken.sol#185,194,200,214,228
Status	Unresolved

Description

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

```
   function excludeFromReward(address account) external
onlyOwner {
        require(!_isExcluded[account], "Account is already
excluded");
        if(_balance_reflected[account] > 0) {
            _balance_total[account] =
tokenFromReflection(_balance_reflected[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    function includeInReward(address account) external
onlyOwner {
        require(_isExcluded[account], "Account is already
included");
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                _excluded[i] = _excluded[_excluded.length - 1];
                _balance_total[account] = 0;
                _isExcluded[account] = false;
                _excluded.pop();
                break;
            }
        }
    }

    function updateRouter(address _router) external onlyOwner {
        uniswapV2Router = IUniswapV2Router(_router);
    }

    function setMaxTransaction (uint256 _maxTransaction)
external onlyOwner {
        require(_maxTransaction >= totalSupply / 1000, "Max
Transaction must be greater than 0.1% of supply");
        maxTransaction = _maxTransaction;
    }

    function manage_excludeFromFee(address[] calldata
addresses, bool status) external onlyOwner {
        for (uint256 i; i < addresses.length; ++i) {
            isFeeExempt[addresses[i]] = status;
        }
```

Recommendation

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

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	DripToken.sol#L200,228,289
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract updates the isFeeExempt status of an account even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
   function updateRouter(address _router) external onlyOwner {
        uniswapV2Router = IUniswapV2Router(_router);
    }

    function _setAllFees(uint256 _comboFee, uint256
_reflectionFee) internal {
        comboFee = _comboFee;
        reflectionFee = _reflectionFee;
    }

    function manage_excludeFromFee(address[] calldata
addresses, bool status) external onlyOwner {
        for (uint256 i; i < addresses.length; ++i) {
            isFeeExempt[addresses[i]] = status;
        }
```

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.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	DripToken.sol#L339
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

Specifically, the contract is utilizing the variables `swapAndLiquifyEnabled` and `swapThreshold` in a conditional statement that triggers the `transferTaxes` function. However, the actual implementation is different from what these variable names suggest, as they are processed to a `transfer` functionality and not to a swap functionality. This discrepancy between the variable names and their actual use can lead to confusion and misunderstandings for developers, auditors, or anyone else reading the code.

```
        if(!inSwapAndLiquify && from != uniswapV2Pair &&
swapAndLiquifyEnabled && balanceOf(address(this)) >
swapThreshold){
             transferTaxes();
         }
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. It is recommended to rename the variables `swapAndLiquifyEnabled` and `swapThreshold` to names that accurately reflect their actual purpose and implementation within the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/DripToken.sol#L62,71,72,74,81,87,94,200,204,214,219,225,291
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);
mapping (address => uint256) public _balance_reflected
mapping (address => uint256) public _balance_total
mapping (address => bool) public _isExcluded
address[] public _excluded
uint256 public _contractReflectionStored = 0
uint256 private _supply_reflected = (MAX - (MAX % totalSupply))
address _router
ITaxContract _taxContract
uint256 _maxTransaction
uint256 _threshold
bool _status

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/DripToken.sol#L266,269
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.

```
tReflection = ( tAmount * reflectionFee ) / (_fee_denominator)
rReflection = tReflection * _getRate()
```

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	contracts/DripToken.sol#L226
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 i
```

Recommendation

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

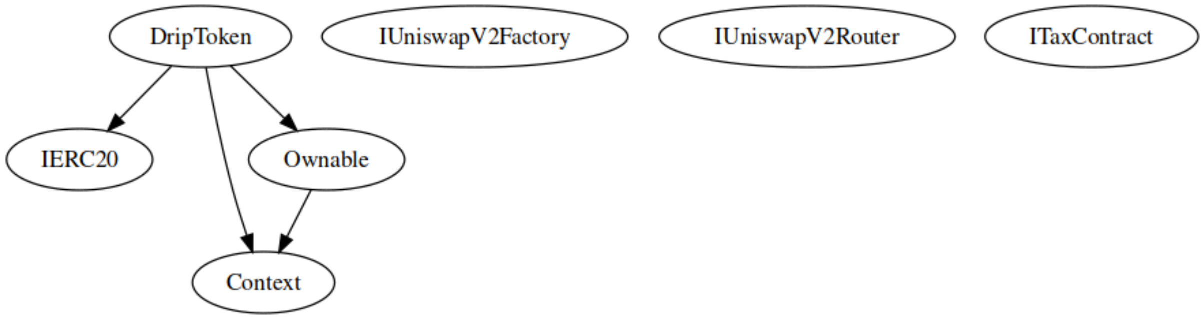
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			

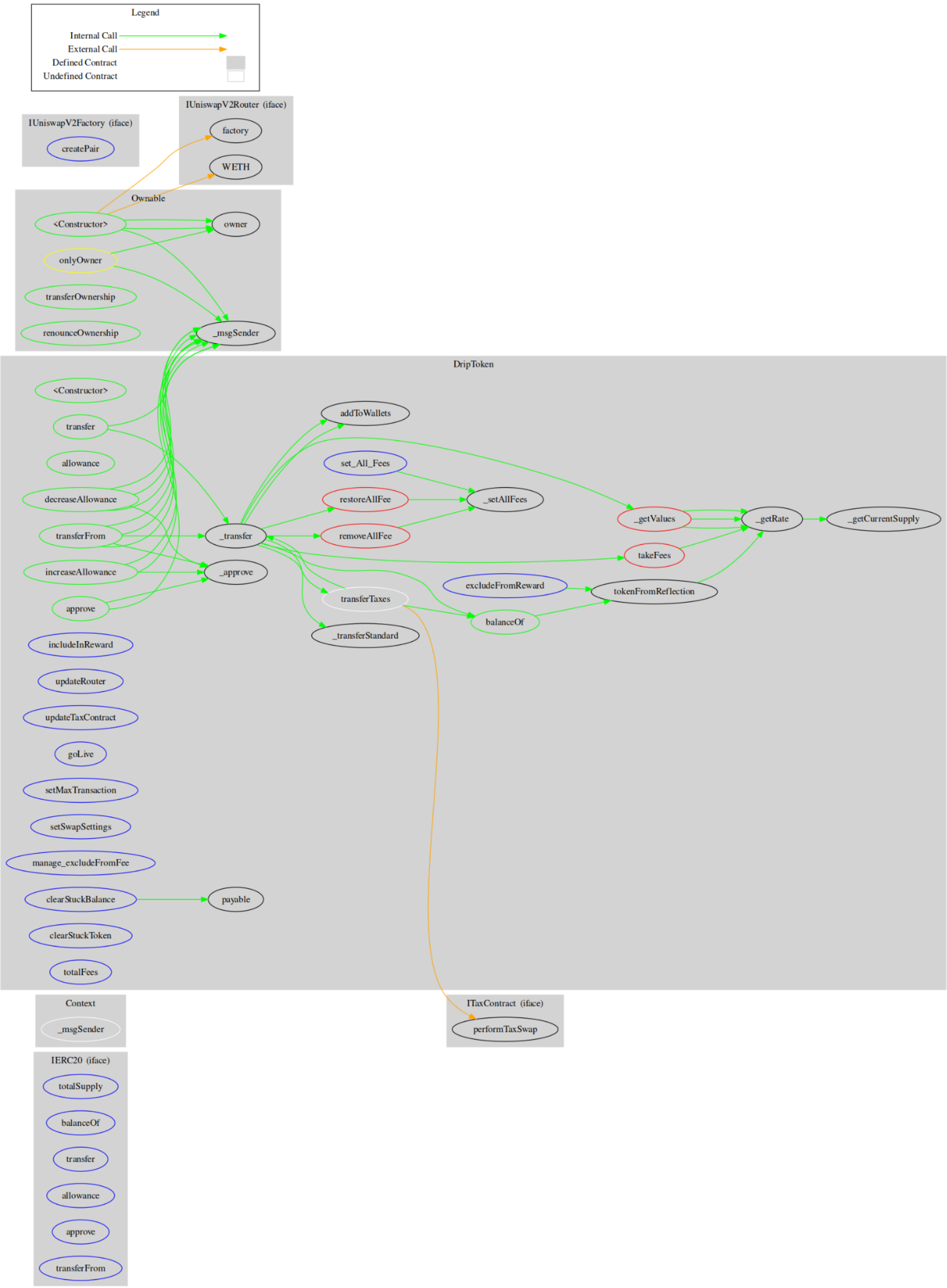
	createPair	External	✓	-
IUniswapV2Router	Interface			
	factory	External		-
	WETH	External		-
ITaxContract	Interface			
	performTaxSwap	External	✓	-
DripToken	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	tokenFromReflection	Public		-
	addToWallets	Internal	✓	
	excludeFromReward	External	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	updateRouter	External	✓	onlyOwner

	updateTaxContract	External	✓	onlyOwner
	goLive	External	✓	onlyOwner
	setMaxTransaction	External	✓	onlyOwner
	setSwapSettings	External	✓	onlyOwner
	manage_excludeFromFee	External	✓	onlyOwner
	clearStuckBalance	External	✓	onlyOwner
	clearStuckToken	External	✓	onlyOwner
	_getRate	Private		
	_getCurrentSupply	Private		
	_getValues	Private		
	takeFees	Private	✓	
	_setAllFees	Internal	✓	
	set_All_Fees	External	✓	onlyOwner
	totalFees	External		-
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	transferTaxes	Internal	✓	
	_approve	Private	✓	
	_transfer	Private	✓	
	_transferStandard	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
createPair
IUniswapV2Router (iface)
factory
WETH
Ownable
<Constructor>
owner
onlyOwner
transferOwnership
renounceOwnership
_msgSender
DripToken
<Constructor>
transfer
allowance
decreaseAllowance
transferFrom
increaseAllowance
approve
includeInReward
updateRouter
updateTaxContract
goLive
setMaxTransaction
setSwapSettings
manage_excludeFromFee
clearStuckBalance
clearStuckToken
totalFees
_transfer
_approve
payable
addToWallets
set_All_Fees
restoreAllFee
removeAllFee
transferTaxes
_transferStandard
_setAllFees
excludeFromReward
balanceOf
_getValues
takeFees
tokenFromReflection
_getRate
_getCurrentSupply
Context
_msgSender
ITaxContract (iface)
performTaxSwap
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

$DRIP 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. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract will eliminate all the contract threats. There is also a limit of max 20% 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