Cyberscope

Audit Report

Payout Coin

May 2023

Network GOERLI

Address 0x4bfc8c7af9aaa83a11af8678b48baa81f906e169

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PVC	Price Volatility Concern	Unresolved
●	PRE	Potential Reentrance Exploit	Unresolved
●	ZD	Zero Division	Unresolved
●	RAI	Redundant Award Iterations	Unresolved
●	RV	Randomization Vulnerability	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Pay
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://goerli.etherscan.io/address/0x4bfc8c7af9aaa83a11af8678b48baa81f906e169
Address	0x4bfc8c7af9aaa83a11af8678b48baa81f906e169
Network	GOERLI
Symbol	Pay
Decimals	9
Total Supply	1,000,000,000

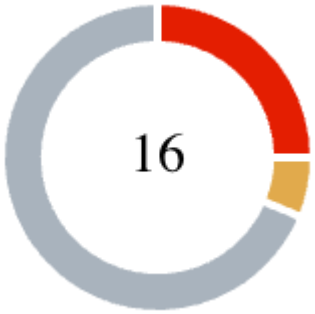
Audit Updates

Initial Audit	28 May 2023

Source Files

Filename	SHA256
Pay.sol	f65f8c3090c8948d6a9799ffff514becba676c8855547ee95a114a29e8fd2a6d

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Critical
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxWalletAmount` to zero. As a result, the contract may operate as a honeypot.

```
if(!_isExcludedFromMaxWallet[recipient])
{
    require(balanceOf(recipient) + amount <= _maxWalletAmount,
"Maximum wallet limit!!");
}
```

Additionally, the contract contains a lot of implementation misconceptions that may lead to a violation of the expected business logic. Read more information on the `PRE` and `ZD` findings.

Recommendation

The contract could embody a check for not allowing setting the `_maxWalletAmount` less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. 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.

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	Pay.sol#L452
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the fee setter functions with a high 10. As a result, the total fees will sum up to 30%.

```
function setMarketingFeePercent(uint256 updatedMarketingFee)
external onlyOwner {
  require(updatedMarketingFee <= 10, "Fee is crossing the
boundaries");
  marketingFee = updatedMarketingFee;
}

function setLotteryFeePercent(uint256 updatedLotteryFee) external
onlyOwner {
  require(updatedLotteryFee <= 10, "Fee is crossing the
boundaries");
  lotteryFee = updatedLotteryFee;
}

function setLiquidityFeePercent(uint256 updatedLiquidityFee)
external onlyOwner {
  require(liquidityFee <= 10, "Fee is crossing the boundaries");
  liquidityFee = updatedLiquidityFee;
}
```

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. 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.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Critical
Location	Pay.sol#L617
Status	Unresolved

Description

As part of the liquidation process, the contract accumulates ETH to the `ethForLottery` state variable. Once the `ethForLottery` reaches a threshold, the amount is transferred pseudo-randomly to one of the holders. The contract also offers the ability to the owner to withdraw the ETH of the contract. If the owner withdraws the contract's ETH, the `ethForLottery` variable is not updated. Additionally, the contract does not reset the `ethForLottery` variable once the reward is sent to the winner. As a result, the expression `address(this).balance - ethForLottery` will produce a subtraction underflow.

```
uint256 ethBalance = address(this).balance - ethForLottery;
...
function withdrawStuckETH() external onlyOwner{
    require (address(this).balance > 0, "Can't withdraw negative or
zero");
    payable(owner()).transfer(address(this).balance);
}
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated. Some suggested actions is:

- Update the `ethForLottery` when the `withdrawStuckETH()` is triggered and once the winner receives the reward.
- Compare the `ethForLottery` with the `address(this).balance` before proceeding to the subtraction.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Pay.sol#L560
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `numTokensSellToAddToLiquidity` 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 ETH.

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

```
bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity;
```

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

PRE - Potential Reentrance Exploit

Criticality	Critical
Location	Pay.sol#L638
Status	Unresolved

Description

The contract makes an external call to transfer funds to recipients using the payable transfer method. The recipient could be a malicious contract that has an untrusted code in its fallback function that makes a recursive call back to the original contract. The re-entrance exploit could be used by a malicious user to drain the contract's funds or to perform unauthorized actions. This could happen because the original contract does not update the state before sending funds.

```
payable(winner).transfer(ethForLottery);
```

Recommendation

The team is advised to prevent the potential re-entrance exploit as part of the solidity best practices. Some suggestions are:

- Add lockers/mutexes in the method scope. It is important to note that mutexes do not prevent cross-function reentrancy attacks.
- Do Not allow contract addresses to receive funds.
- Proceed with the external call as the last statement of the method, so that the state will have been updated properly during the re-entrance phase.

ZD - Zero Division

Criticality	Critical
Location	Pay.sol#L600
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

If the sum of `marketingFee + liquidityFee + lotteryFee` is zero, then the `totalFee()` will yield a zero number and produce a zero division.

```
uint256 tokensToLP = (tokensToLiquify * liquidityFee / totalFee())
/ 2 ;
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

RAI - Redundant Award Iterations

Criticality	Medium
Location	Pay.sol#L628
Status	Unresolved

Description

The contract iterates the entire holders' array in order to match the winner. Even if the winner is in the first position of the array, the algorithm will iterate the whole structure. This procedure increases gas consumption dramatically.

```
for(uint256 i = 0; i < holders.length; i++)
{
    if( i == randomNum )
    {
        winner = payable(holders[i]);
    }
}
payable(winner).transfer(ethForLottery);
```

Recommendation

The team is advised to revisit the for-loop statement in order to improve gas consumption. Since the random number range is between the holders' array boundaries, then the number could be used directly as index in the array. As a result, the gas consumption complexity will be decreased from O(n) to O(1).

RV - Randomization Vulnerability

Criticality	Minor / Informative
Location	Pay.sol#L673
Status	Unresolved

Description

The contract is using an on-chain technique in order to determine random numbers. The blockchain runtime environment is fully deterministic, as a result, the pseudo-random numbers could be predicted.

```
    function random(uint number) public view returns(uint){
        return
uint(keccak256(abi.encodePacked(block.timestamp,block.difficulty,
        msg.sender))) % number;
    }
```

Recommendation

The contract could use an advanced randomization technique that guarantees an acceptable randomization factor. For instance, the Chainlink VRF (Verifiable Random Function). https://docs.chain.link/docs/chainlink-vrf

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Pay.sol#L626
Status	Unresolved

Description

The contract sends funds to a `marketingAddress` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to 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.

```
payable(marketingAddress).transfer(ethForMarketing);
```

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

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Pay.sol#L358
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.

```
_totalSupply
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Pay.sol#L135,136,153,175,314,315,316,317,318,322,339,472,477,693
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 (uint);
function WETH() external pure returns (address);
mapping (address => uint) internal _balances
mapping (address => mapping (address => uint)) internal _allowances
mapping (address => bool) public _isExcludedFromFee
mapping (address => bool) public AMMs
mapping (address => bool) public _isExcludedFromMaxWallet
uint256 internal _totalSupply
uint256 public _maxWalletAmount = 2500000 * 10**9
bool _enabled
uint256 _numTokensSellToAddToLiquidity
address _address
```

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	Pay.sol#L450,456,463,479,511,516
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.

```
marketingFee = updatedMarketingFee
lotteryFee = updatedLotteryFee
liquidityFee = updatedLiquidityFee
numTokensSellToAddToLiquidity = _numTokensSellToAddToLiquidity
_maxWalletAmount = maxWalletAmount
lotteryReward = amount
```

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	Pay.sol#L645
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 swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

...
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Pay.sol#L469
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.

```
marketingAddress = 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	Pay.sol#L3
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 ^0.8.20;
```

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	Pay.sol#L697
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(owner(),
IERC20(_address).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.

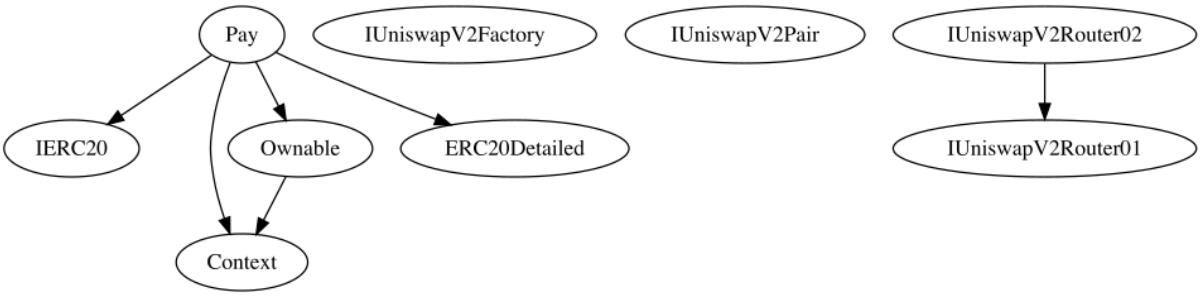
Functions Analysis

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

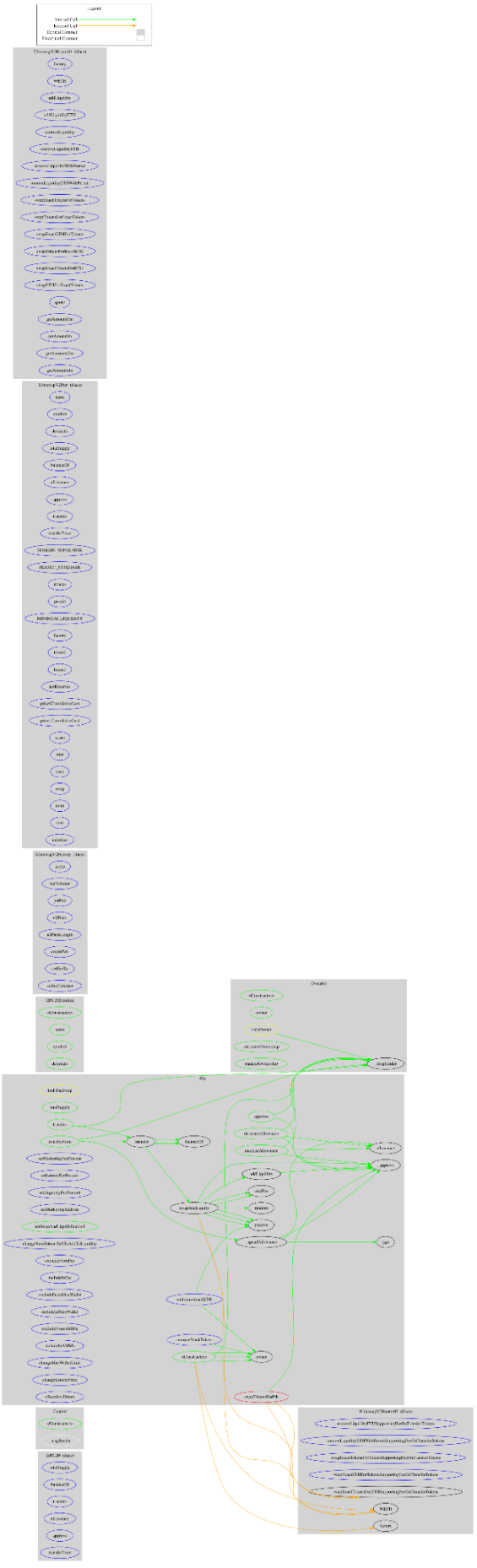
ERC20Detailed	Implementation			
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
Pay	Implementation	Context, Ownable, IERC20, ERC20Detailed		
		Public	✓	ERC20Detailed
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	approve	Public	✓	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_spendAllowance	Internal	✓	
	setMarketingFeePercent	External	✓	onlyOwner
	setLotteryFeePercent	External	✓	onlyOwner
	setLiquidityFeePercent	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner

	changeNumTokensSellToAddToLiquidity	External	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	includeInFee	External	✓	onlyOwner
	excludeFromMaxWallet	External	✓	onlyOwner
	includeInMaxWallet	External	✓	onlyOwner
	excludeFromAMMs	External	✓	onlyOwner
	includeInAMMs	External	✓	onlyOwner
	changeMaxWalletLimit	External	✓	onlyOwner
	changeLotteryPrize	External	✓	onlyOwner
		External	Payable	-
	_transfer	Internal	✓	
	totalFee	Internal		
	swapAndLiquify	Private	✓	
	random	Public		-
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_approve	Internal	✓	
	withdrawStuckETH	External	✓	onlyOwner
	removeStuckToken	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Payout Coin 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, manipulate the fees and transfer funds to the team's wallet. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. There is also a limit of max 30% 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