Cyberscope

Audit Report

Make Trump President Again

December 2023

Network ETH

Address 0x961b1eae74fcd6a3266099860c3d8af9c0bf4526

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ITH	Inefficient Transaction Handling	Unresolved
●	RFU	Redundant Flag Usage	Unresolved
●	RCS	Redundant Code Segments	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	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
ELFM - Exceeds Fees Limit 7
Description 7
Recommendation 7
ITH - Inefficient Transaction Handling 8
Description 8
Recommendation 9
RFU - Redundant Flag Usage 10
Description 10
Recommendation 11
RCS - Redundant Code Segments 12
Description 12
Recommendation 12
IDI - Immutable Declaration Improvement 14
Description 14
Recommendation 14
L02 - State Variables could be Declared Constant 15
Description 15
Recommendation 15
L04 - Conformance to Solidity Naming Conventions 16
Description 16
Recommendation 16
L05 - Unused State Variable 18
Description 18
Recommendation 18
L09 - Dead Code Elimination 19
Description 19
Recommendation 19
L19 - Stable Compiler Version 21
Description 21
Recommendation 21
Functions Analysis 22
Inheritance Graph 25

Flow Graph **26**
Summary **27**
Disclaimer **28**
About Cyberscope **29**

Review

Contract Name	TrumpToken
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization	200 runs
Explorer	https://etherscan.io/address/0x961b1eae74fcd6a3266099860c3d8af9c0bf4526
Address	0x961b1eae74fcd6a3266099860c3d8af9c0bf4526
Network	ETH
Symbol	Trump42024
Decimals	18
Total Supply	1,000,000,000

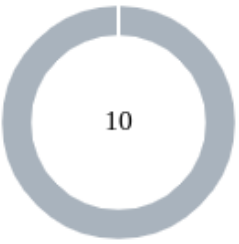
Audit Updates

Initial Audit	13 Dec 2023

Source Files

Filename	SHA256
TrumpToken.sol	e5d4da455a0e09c47e69e8a8ad1b97280839707c4e5e8ac00d5794771eb18656

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	TrumpToken.sol#L572
Status	Unresolved

Description

The contract is currently structured to impose transaction fees that exceed the allowed limit of 25%. Notably, for the first 100 transactions, the contract imposes a 30% fee. These fee structures are significantly higher than the standard permissible limit of 25% for transaction fees in smart contracts.

```
function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        uint256 taxAmount = 0;
        Flag flag = Flag.None;
        _cnt_tx += 1;
        uint _fee;
        if (block.number < 2 + _fisrt_block_num) {
            _fee = 70;
        } else if (_cnt_tx < 100) {
            _fee = 30;
        } else if (_cnt_tx < 150) {
            _fee = 15;
          ...
      }
```

Recommendation

It is recommended to adjust the fee structure within the contract to align with standard practices and ensure fairness in transactions. Specifically, the fees should be capped at a maximum of 25% to adhere to common limits and maintain user trust. Reducing the 30% fee for the first 100 transactions to a maximum of 25% would make the contract more trustworthy to users.

ITH - Inefficient Transaction Handling

Criticality	Minor / Informative
Location	TrumpToken.sol#L563
Status	Unresolved

Description

The contract's `_transfer` function is structured to handle transaction fees and count transactions. However, the `_cnt_tx` variable, which tracks the number of transactions, is incremented with every transfer. This variable is only relevant for determining the fee rate for the first `150` transactions. Beyond this threshold, the `_cnt_tx` increment becomes redundant, as it no longer influences the transaction fee calculation. This unnecessary incrementation of `_cnt_tx` after the first `150` transactions adds an avoidable overhead to the contract's operations.

Additionally, the contract executes a transfer to the `_taxWallet` regardless of whether the `taxAmount` is zero. In scenarios where `taxAmount` equals zero (which occurs when the `_fee` is 0), the contract still performs a transfer to the `_taxWallet`. This action results in an unnecessary consumption of gas, as it executes a transaction that has no effect on the state of the contract or the balances of the addresses involved.

```
function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        uint256 taxAmount = 0;
        Flag flag = Flag.None;
        _cnt_tx += 1;
        uint _fee;

        ...

        super._transfer(from, to, amount - taxAmount);
        super._transfer(from, _taxWallet, taxAmount);
    }
```

`

Recommendation

It is recommended to optimize the `_transfer` function to address these inefficiencies. For the `_cnt_tx` variable, a conditional check can be introduced to stop incrementing it once it reaches 150. This change will prevent unnecessary computation and storage operations, thereby optimizing contract performance. Regarding the `taxAmount`, it is advisable to add a condition that bypasses the transfer to `_taxWallet` when `taxAmount` is zero. This will avoid the unnecessary gas expenditure associated with these redundant transactions. Implementing these optimizations will enhance the contract's efficiency, reduce gas costs, and streamline its execution.

RFU - Redundant Flag Usage

Criticality	Minor / Informative
Location	TrumpToken.sol#L541,584
Status	Unresolved

Description

The contract contains the enumeration (enum) `Flag`, which includes the values `None`, `Sell`, and `Buy`. This `enum` is used within the `_transfer` function to set the `flag` variable based on the nature of the transaction, whether it is a sell or a buy. The flag variable is set to `Flag.Sell` if the to address is the `swapPair`, and to `Flag.Buy` if the from address is the `swapPair`. However, this Flag enum, and the flag variable, are not utilized in any subsequent code segments or logic. Their declaration and assignment do not influence or alter the contract's behavior in any observable way. This lack of utilization renders the Flag enum and the flag variable redundant, contributing no functional value to the contract's operations.

```
  enum Flag {
        None,
        Sell,
        Buy
    }
    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        uint256 taxAmount = 0;
        Flag flag = Flag.None;
        ...

        if (to == swapPair) {
            taxAmount = amount * _fee / 100;
            flag = Flag.Sell;
        } else if (from == swapPair) {
            taxAmount = amount * _fee / 100;
            flag = Flag.Buy;
        }
...}
```

Recommendation

It is recommended to remove the `Flag` `enum` and the `flag` variable from the contract to optimize its efficiency and clarity. Eliminating these redundant elements will streamline the contract, reducing unnecessary complexity and potential confusion. This simplification can also contribute to reduced gas costs, as every operation in a smart contract on blockchain networks incurs a cost. By removing unused code, the contract becomes more efficient in terms of execution and easier to maintain and audit. If the intention was to use the Flag enum for future features or logic, it is advisable to reintroduce it when it has a defined role in the contract's functionality.

RCS - Redundant Code Segments

Criticality	Minor / Informative
Location	TrumpToken.sol#L528,594,608
Status	Unresolved

Description

The contract contains the `_maxTokens` variable, and the `setMinTokens` and `_addLiquidity` functions. The `_maxTokens` variable, despite being declared and having a setter method through `setMinTokens`, is not used in the contract's logic or functionality. Similarly, the `_addLiquidity` function, while implemented, is not utilized into the contract's core processes or transactional mechanisms. This lack of meaningful integration and application renders this variable and the functions redundant, contributing no substantial functionality to the contract.

```
    uint256 public _maxTokens = 100 * 10 ** decimals();

    function _addLiquidity(
        uint256 tokenAmount,
        uint256 ethAmount
    ) private {
        swapRouter.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0,
            0,
            owner(),
            block.timestamp
        );
    }

    function setMinTokens(uint256 newValue) external onlyOwner {
        _maxTokens = newValue * 10 ** decimals();
    }
```

Recommendation

It is recommended to streamline the contract by removing these redundant code segments. Renoving the `_maxTokens` variable, the `setMinTokens` function, and the `_addLiquidity` function will not only simplify the contract but also enhance its clarity and efficiency. This action will reduce unnecessary complexity and potential confusion, making the contract more straightforward for users and auditors to understand and interact with. Additionally, removing unused code can slightly reduce the contract's deployment and execution costs, contributing to overall optimization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	TrumpToken.sol#L554
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.

```
swapPair
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	TrumpToken.sol#L517,532,533,534
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.

```
address public _taxWallet = 0x85CF0edd7FDC95D92D208b157886B0583aFd4A87
uint256 private _tempReward = 0
uint256 private _tempMK = 0
uint256 private _tempLP = 0
```

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	TrumpToken.sol#L98,517,521,524,528
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);
address public _taxWallet = 0x85CF0edd7FDC95D92D208b157886B0583aFd4A87
uint256 private _fisrt_block_num = block.number
uint256 private _cnt_tx = 0
uint256 public _maxTokens = 100 * 10 ** decimals()
```

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	TrumpToken.sol#L532,533,534
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.

```
uint256 private _tempReward = 0
uint256 private _tempMK = 0
uint256 private _tempLP = 0
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	TrumpToken.sol#L425,594
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");

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds
balance");
        unchecked {
            _balances[account] = accountBalance - amount;
            // Overflow not possible: amount <= accountBalance <=
totalSupply.
            _totalSupply -= 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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TrumpToken.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 ^0.8.0;
```

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.

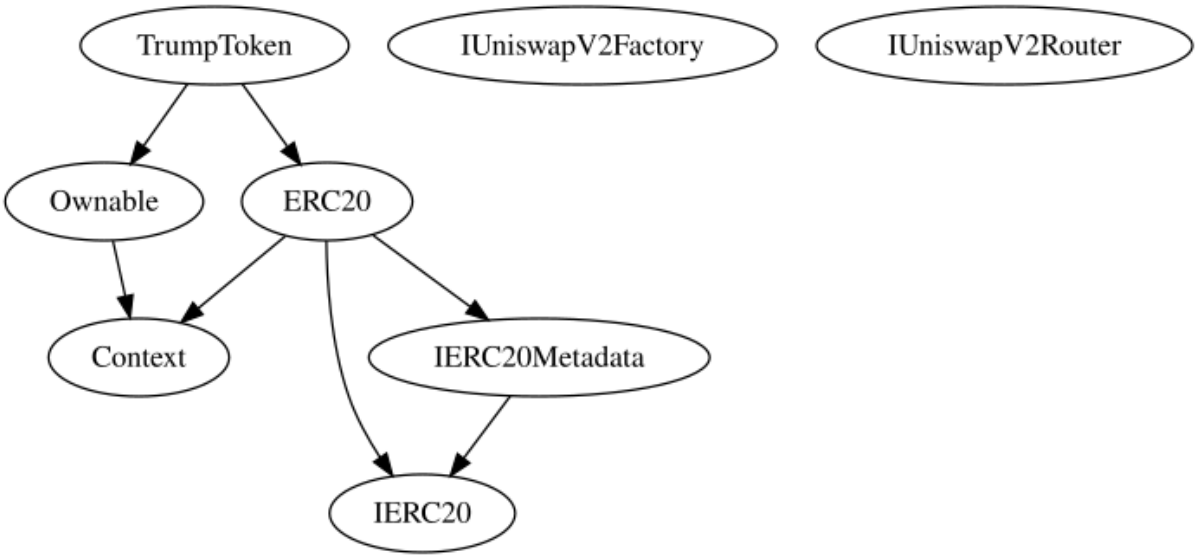
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-

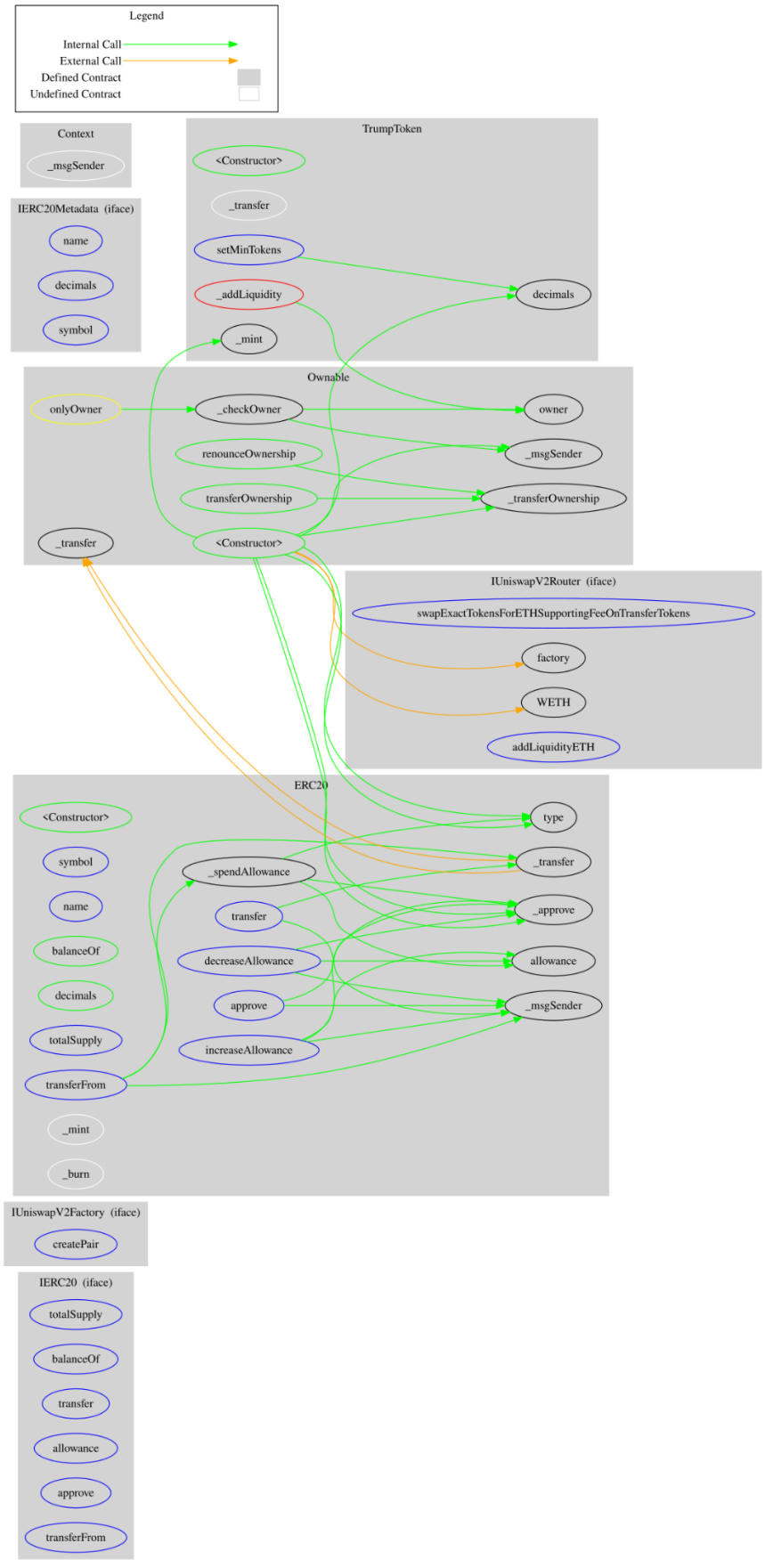
IERC20Metadata	Interface	IERC20		
	name	External		-
	decimals	External		-
	symbol	External		-
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	symbol	External		-
	name	External		-
	balanceOf	Public		-
	decimals	Public		-

	totalSupply	External		-
	allowance	Public		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
	decreaseAllowance	External	✓	-
	increaseAllowance	External	✓	-
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_transfer	Internal	✓	
TrumpToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	_transfer	Internal	✓	
	_addLiquidity	Private	✓	
	setMinTokens	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Context
_msgSender
IERC20Metadata (iface)
name
decimals
symbol
TrumpToken
<Constructor>
_transfer
setMinTokens
_addLiquidity
_mint
decimals
Ownable
onlyOwner
_checkOwner
owner
renounceOwnership
_msgSender
transferOwnership
_transferOwnership
_transfer
<Constructor>
IUniswapV2Router (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
factory
WETH
addLiquidityETH
ERC20
<Constructor>
symbol
name
balanceOf
decimals
totalSupply
transferFrom
_mint
_burn
_spendAllowance
transfer
decreaseAllowance
approve
increaseAllowance
type
_transfer
_approve
allowance
_msgSender
IUniswapV2Factory (iface)
createPair
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Make Trump President Again contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Make Trump President Again 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 implements a dynamic fee structure based on certain conditions, which are primarily determined by the count of transactions (`_cnt_tx`). The fee is set according to the following rules:

- 30% fee for the first 100 transactions
- 15% fee if the transaction count is between 100 and 149
- 5 % fee for all subsequent 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