Cyberscope

Audit Report

VEEPE

August 2024

Network BASE

Address 0x637a2d329bca4661fa6741d0aa555d742e3798de

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	SemiResolved
●	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
	CO	Code Optimization	Unresolved
	HV	Hardcoded Values	SemiResolved
	IDI	Immutable Declaration Improvement	Unresolved
	MEE	Missing Events Emission	SemiResolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	RRA	Redundant Repeated Approvals	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	VEEPETOKEN
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Testing Deploy	https://testnet.bscscan.com/address/0x4f1be254956ca6ab7e7bec9207f0b119270ddd46
Explorer	https://basescan.org/address/0x637a2d329bca4661fa6741d0aa555d742e3798de
Address	0x637a2d329bca4661fa6741d0aa555d742e3798de
Network	BASE
Symbol	VEEPE
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

Audit Updates

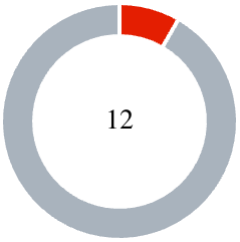
Initial Audit	23 Aug 2024
Corrected Phase 2	08 Aug 2024

Source Files

Filename	SHA256

src/VEEPETOKEN.sol	ffc1e1e74060a8b3401cc2712e0aa85648d68d0342d11f553a4f455fed2ca093

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	1
Medium	0	0	0	0
Minor / Informative	9	0	0	2

ST - Stops Transactions

Criticality	Critical
Location	src/VEEPETOKEN.sol#L453,513
Status	SemiResolved

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. Additionally, the contract owner can set a relatively large number as deadBlocks, so the users will be taxed with 90% fees for a relatively long period.

```
function enableTrading(uint256 deadBlocks) external onlyOwner {
    require(!tradingActive, "Cannot reenable trading");
    tradingActive = true;
    swapEnabled = true;
    tradingActiveBlock = block.number;
    blockForPenaltyEnd = tradingActiveBlock + deadBlocks;
    emit EnabledTrading();
}
...
if(!tradingActive){
    require(_isExcludedFromFees[from] || _isExcludedFromFees[to],
"Trading is not active.");
}
```

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.

The team is also adviced to add a limit in the maximum amount of `deadBlocks` .

Team Update

The team removed the `deadblocks` .

CO - Code Optimization

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L410,415,557,566,574,634,654
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 check for `buyTotalFees > 0` and `sellTotalFees > 0` is redundant since `buyTotalFees` and `sellTotalFees` cannot be zero.

```
buyMarketingFee = 10;
buyDevFee = 10;
buyBurnFee = 0;
buyTotalFees = buyMarketingFee + buyDevFee + buyBurnFee;
sellMarketingFee = 10; sellDevFee = 15;
sellBurnFee = 5;
sellTotalFees = sellMarketingFee + sellDevFee + sellBurnFee;
...
if(earlyBuyPenaltyInEffect() && automatedMarketMakerPairs[from] &&
!automatedMarketMakerPairs[to] && buyTotalFees > 0){
...
else if (automatedMarketMakerPairs[to] && sellTotalFees > 0){
...
else if(automatedMarketMakerPairs[from] && buyTotalFees > 0) {
```

At `swapBack()` the variable `tokensForBurn` is zeroed out twice unnecessarily.

```
function swapBack() private {

    if(tokensForBurn > 0 && balanceOf(address(this)) >=
tokensForBurn) {
        _burn(address(this), tokensForBurn);
    }
    tokensForBurn = 0;

    uint256 contractBalance = balanceOf(address(this));
    uint256 totalTokensToSwap = tokensForMarketing + tokensForDev;

    if(contractBalance == 0 || totalTokensToSwap == 0) {return;}

    if(contractBalance > swapTokensAtAmount * 20){
        contractBalance = swapTokensAtAmount * 20;
    }

    bool success;

    swapTokensForEth(contractBalance);

    uint256 ethBalance = address(this).balance;
    uint256 ethForDev = ethBalance * tokensForDev /
totalTokensToSwap;

    tokensForMarketing = 0;
    tokensForDev = 0;
    tokensForBurn = 0;

    (success,) = address(devAddress).call{value: ethForDev}("");

    (success,) = address(marketingAddress).call{value:
address(this).balance}("");
}
```

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.

HV - Hardcoded Values

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L400,559,567,575,641
Status	SemiResolved

Description

The contract contains several instances where numeric values are directly hardcoded into the logic instead of being assigned to constant variables with meaningful names. Hardcoding values can lead to various issues, such as reduced code readability, increased likelihood of introducing errors during maintenance, and difficulty in managing and updating values consistently throughout the contract.

```
uint256 totalSupply = 1 * 1e9 * 1e18;

maxBuyAmount = totalSupply * 1 / 100;
maxSellAmount = totalSupply * 1 / 100;
maxWalletAmount = totalSupply * 2 / 100;
swapTokensAtAmount = totalSupply * 5 / 10000;
...
fees = amount * 90 / 100;
...
fees = amount * sellTotalFees / 1000;
...
fees = amount * buyTotalFees / 1000;
...
if(contractBalance > swapTokensAtAmount * 20){
    contractBalance = swapTokensAtAmount * 20;
}
```

Recommendation

The team is advised to replace hardcoded numeric values with constant variables that have meaningful names. This will enhance code readability and maintainability, and reduce the potential for errors during updates.

Team Update

The team removed the `maxBuyAmount` , `maxSellAmount` and `maxWalletAmount` .

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L358,364,365,366,367,369,370,371,372
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.

```
lpPair
buyMarketingFee
buyDevFee
buyBurnFee
buyTotalFees
sellMarketingFee
sellDevFee
sellBurnFee
sellTotalFees
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L484,669,674
Status	SemiResolved

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.

```
_isExcludedMaxTransactionAmount[updAds] = isEx;
marketingAddress = payable(_marketingAddress);
devAddress = payable(_devAddress);
```

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.

Team Update

The team removed the `excludeFromMaxTransaction` function.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L651
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
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] = dexRouter.WETH();

    _approve(address(this), address(dexRouter), tokenAmount);

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

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L614
Status	Unresolved

Description

The contract is designed to approve token transfers during the contract's operation by calling the `_approve` function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
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] = dexRouter.WETH();

    _approve(address(this), address(dexRouter), tokenAmount);

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

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L272,559,564
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.

```
unction WETH() external pure returns (address);

ddress _marketingAddress)
ddress _devAddress)
```

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	src/VEEPETOKEN.sol#L411
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.

```
wapTokensAtAmount = newAmount;
```

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	src/VEEPETOKEN.sol#L507
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.

```
unction addLiquidity(uint256 tokenAmount, uint256 ethAmount)
private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(dexRouter), tokenAmount);

        // add the liquidity
        dexRouter.addLiquidityETH{value: ethAmount}(
...
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            address(0xdead),
            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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L466,467,468,469,473,474,475,476
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.

```
ees = amount * buyTotalFees / 1000;
okensForMarketing += fees * buyMarketingFee / buyTotalFees;
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	src/VEEPETOKEN.sol#L361
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
int256 totalSupply = 1 * 1e9 * 1e18;
```

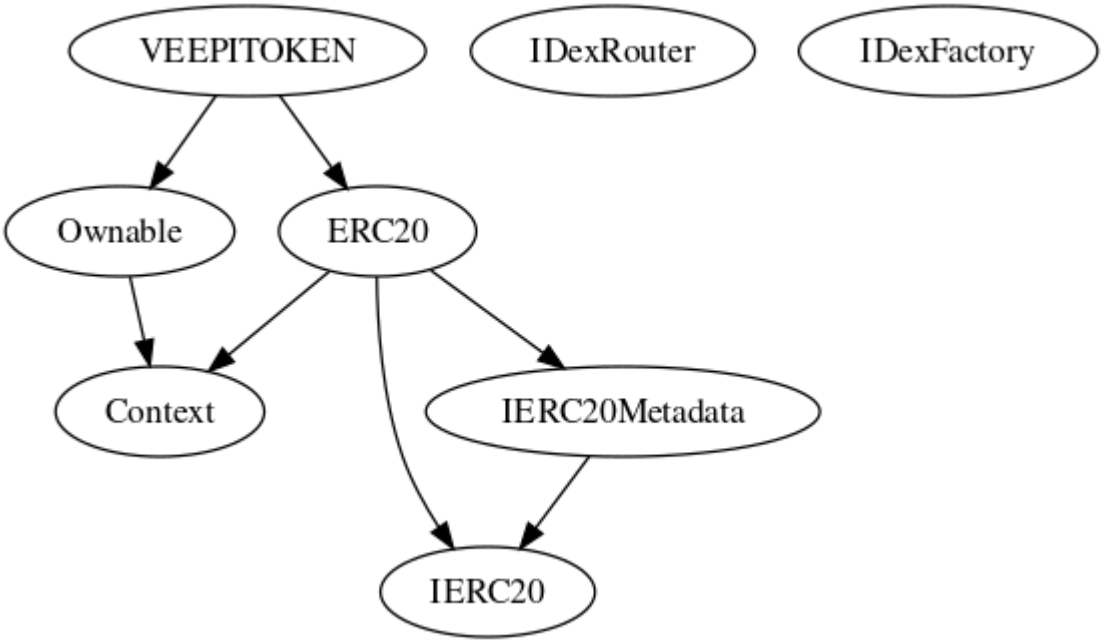
Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

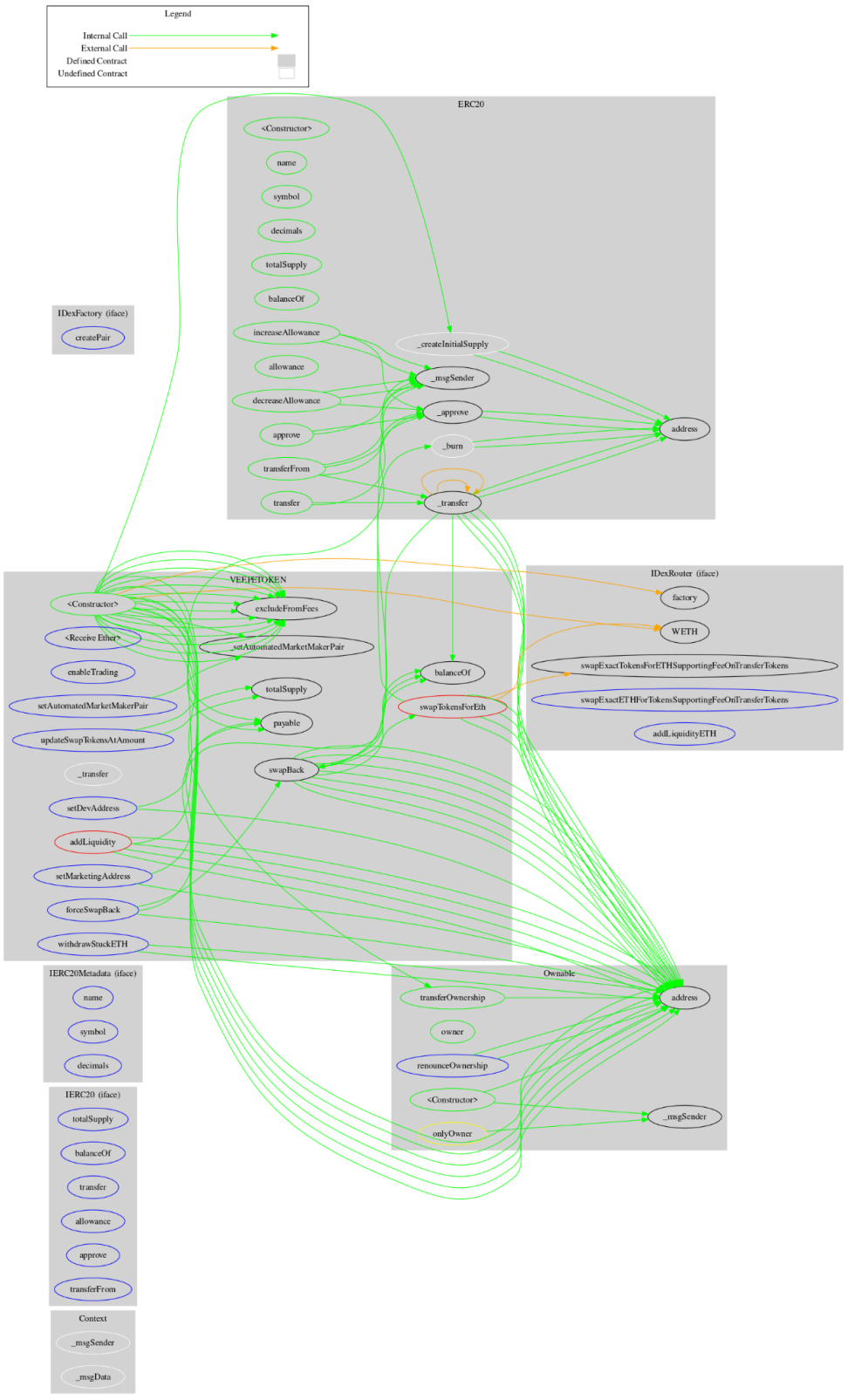
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
VEEPETOKEN	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromFees	Public	✓	onlyOwner
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	
	withdrawStuckETH	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setDevAddress	External	✓	onlyOwner
	forceSwapBack	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
increaseAllowance
allowance
decreaseAllowance
approve
transferFrom
transfer
_createInitialSupply
_msgSender
_approve
_burn
_transfer
address
IDexFactory (iface)
createPair
VEEPETOKEN
<Constructor>
<Receive Ether>
enableTrading
setAutomatedMarketMakerPair
updateSwapTokensAtAmount
_transfer
setDevAddress
addLiquidity
setMarketingAddress
forceSwapBack
withdrawStuckETH
excludeFromFees
_setAutomatedMarketMakerPair
totalSupply
payable
swapBack
balanceOf
swapTokensForEth
IDexRouter (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
addLiquidityETH
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
Ownable
transferOwnership
owner
renounceOwnership
<Constructor>
onlyOwner
address
_msgSender

Summary

VEEPE is an interesting project that has a friendly and growing community. This audit investigates security issues, business logic concerns and potential improvements.

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

cyberscope.io