Cyberscope

Audit Report

Kitten Token

November 2024

Repository : https://github.com/kittentoken/kittentoken/tree/main

Commit : 981c5b87ca30f784fb290e1b7b5b69caaf0b35b3

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	Unresolved

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	PGA	Potential Griefing Attack	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	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

Repository	https://github.com/kittentoken/kittentoken/tree/main
Commit	981c5b87ca30f784fb290e1b7b5b69caaf0b35b3
Badge Eligibility	Must Fix Criticals

Test Deploy	https://sepolia.etherscan.io/address/0x2099EBCdFEF6dBd59b3E13AC1D7EE29Ae25C0f9C

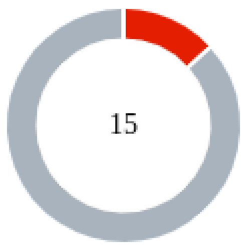
Audit Updates

Initial Audit	15 Nov 2024

Source Files

Filename	SHA256
CoinToken.sol	d2bf17b28b3cd8e35011688da48daecd04f2653fcc9369c2ad52ae7b8efcb7ef

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	13

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	13	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	CoinToken.sol#L219
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.

```
function openTrading() external onlyOwner() {
    if(isTradingOpen) revert TradingIsAlreadyOpen();
    isTradingOpen = true;
    swapEnabled = true;
    emit TradingOpened(block.timestamp);
}
```

In addition, the contract limits transfers with amounts of less than `1000`. Given however that the total supply is dynamically passed by the owner during deployment, `1000` may be a significant value. Therefore, prevent tokens from being transfered.

```
if (amount < 1000) revert ToSmallOrToLargeTransactionAmount();
```

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.

In addition, limits on transactions, if applicable, should be proportional to the total supply of the token and not fixed to an arbitrary value.

BC - Blacklists Addresses

Criticality	Critical
Location	CoinToken.sol#L444,450
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `blacklistAddress` function.

```
function addBots(address[] memory _bots) external onlyOwner {
    for (uint i = 0; i < _bots.length; i++) {
        bots[_bots[i]] = true;
    }
}

function delBots(address[] memory _notBot) external onlyOwner {
    for (uint i = 0; i < _notBot.length; i++) {
        bots[_notBot[i]] = false;
    }
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	CoinToken.sol#L211,219,392,397,406,414,426,432,438,444,450,465,471,476,489
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function addLqToUniswap() external onlyOwner() {}
function openTrading() external onlyOwner() {}
function clearStuckToken(address tokenAddress, uint256 tokens)
external onlyOwner() returns(bool) {}
function manualSwap() external onlyOwner() returns(uint256,
uint256, uint256, uint256) {}
function setFeeWallets(address newDevWalletAddress, address
newMarketingWalletAddress, address newCharityWallet) external
onlyOwner() {}
function setFeeExclusionForAccount(address account, bool boolValue)
external onlyOwner() {}
function setExclusionFromMaxTransaction(address account, bool
boolValue) external onlyOwner() {}
function setAutomatedMarketMakerPair(address pair, bool boolValue)
external onlyOwner() {}
function addBots(address[] memory _bots) external onlyOwner {}
function delBots(address[] memory _notBot) external onlyOwner {}
function removeLimits() external onlyOwner {}
function setSwapTokensAtAmountSupplyPercentage(uint8
newSwapTokensAtAmountSupplyPercentage) external onlyOwner {}
function setSwapPossibility(bool boolValue) external onlyOwner {}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	CoinToken.sol#L118
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.

```
uniswapV2Pair
```

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.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	CoinToken.sol#L476
Status	Unresolved

Description

The contract is not fully conforming to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However the contract implements, a conditional check that prohibits transfers of 0 values.

This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
if (amount < 1000) revert ToSmallOrToLargeTransactionAmount();
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	CoinToken.sol#L438
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function setAutomatedMarketMakerPair(address pair, bool boolValue)
external onlyOwner() {
    if(pair == uniswapV2Pair || pair == ZERO_ADDRESS) revert
FailedSetter();
    automatedMarketMakerPairs[pair] = boolValue;
    emit SetAutomatedMarketMakerPair(pair, boolValue);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

PGA - Potential Griefing Attack

Criticality	Minor / Informative
Location	CoinToken.sol#L250
Status	Unresolved

Description

The `delayTransactionCheck` function includes a conditional statement intended to impose a maximum of 1 transaction per user per block. This opens up the potential for a griefieng attack. During such an incident, a malicious actor could front-run normal sales with spam transactions to prevent legitimate transactions from taking place.

```
function delayTransactionCheck(address from, address to) private {
if (transferDelayEnabled && !isExcludedFromFees[from] &&
!isExcludedFromFees[to] && to != address(uniswapV2Router) &&
!automatedMarketMakerPairs[to] &&
!isExcludedFromMaxTransactionAmount[to]) {
if(tokenHolderLastTransferBlockNumber[tx.origin] >= block.number ||
tokenHolderLastTransferBlockNumber[to] >= block.number) {
revert TransferDelayTryAgainLater();
}
tokenHolderLastTransferBlockNumber[tx.origin] = block.number;
tokenHolderLastTransferBlockNumber[to]        = block.number;
}
}
```

Recommendation

The team is advised to revise the transfer mechanism to ensure all legitimate transactions are processed according to the intended behavior.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	CoinToken.sol#L373
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 {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router), tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        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.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	CoinToken.sol#L117
Status	Unresolved

Description

The contract includes a method that allows the owner to pass the router address as an argument during deployment. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
uniswapV2Router = IUniswapV2Router02(_routerAddress);
```

Recommendation

The team should consider including the router address as a constant immutable value in the codebase.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	CoinToken.sol#L348
Status	Unresolved

Description

The contract sends funds to a `marketingWallet`, a `devWallet` and a `charityWallet` 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.

```
if(marketingFee > 0) marketingWallet.call{value: (amountEthSwapped
* marketingFee) / totalETHFee}("");
if(devFee > 0)       devWallet.call{value: (amountEthSwapped *
devFee) / totalETHFee}("");
if(charityFee > 0)   charityWallet.call{value: (amountEthSwapped *
charityFee) / totalETHFee}("");
```

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.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	CoinToken.sol#L377
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 {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router), tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        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	CoinToken.sol#L52,53,444,450
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.

```
uint8 private constant maxTotalBuyFee  = 50
uint8 private constant maxTotalSellFee = 50
address[] memory _bots
address[] memory _notBot
```

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/stable/style-guide.html#naming-conventions.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CoinToken.sol#L338,353
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.

```
swapTokensAtAmount = (totalSupply() *
swapTokensAtAmountTotalSupplyPercentage) / 1000
uint256 tokenAmountToLiquify = dynamicLiquidityFee > 0 ?
(swapTokensAtAmount * dynamicLiquidityFee) / totalFee / 2: 0
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CoinToken.sol#L129,130,131
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.

```
devWallet       = payable(_devWallet)
marketingWallet = payable(_marketingWallet)
charityWallet   = payable(_charityWallet)
```

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	CoinToken.sol#L20
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 ^0.8.4;
```

Recommendation

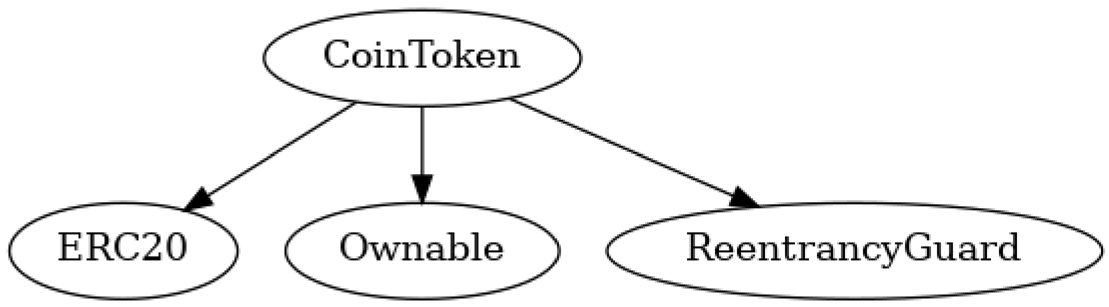
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

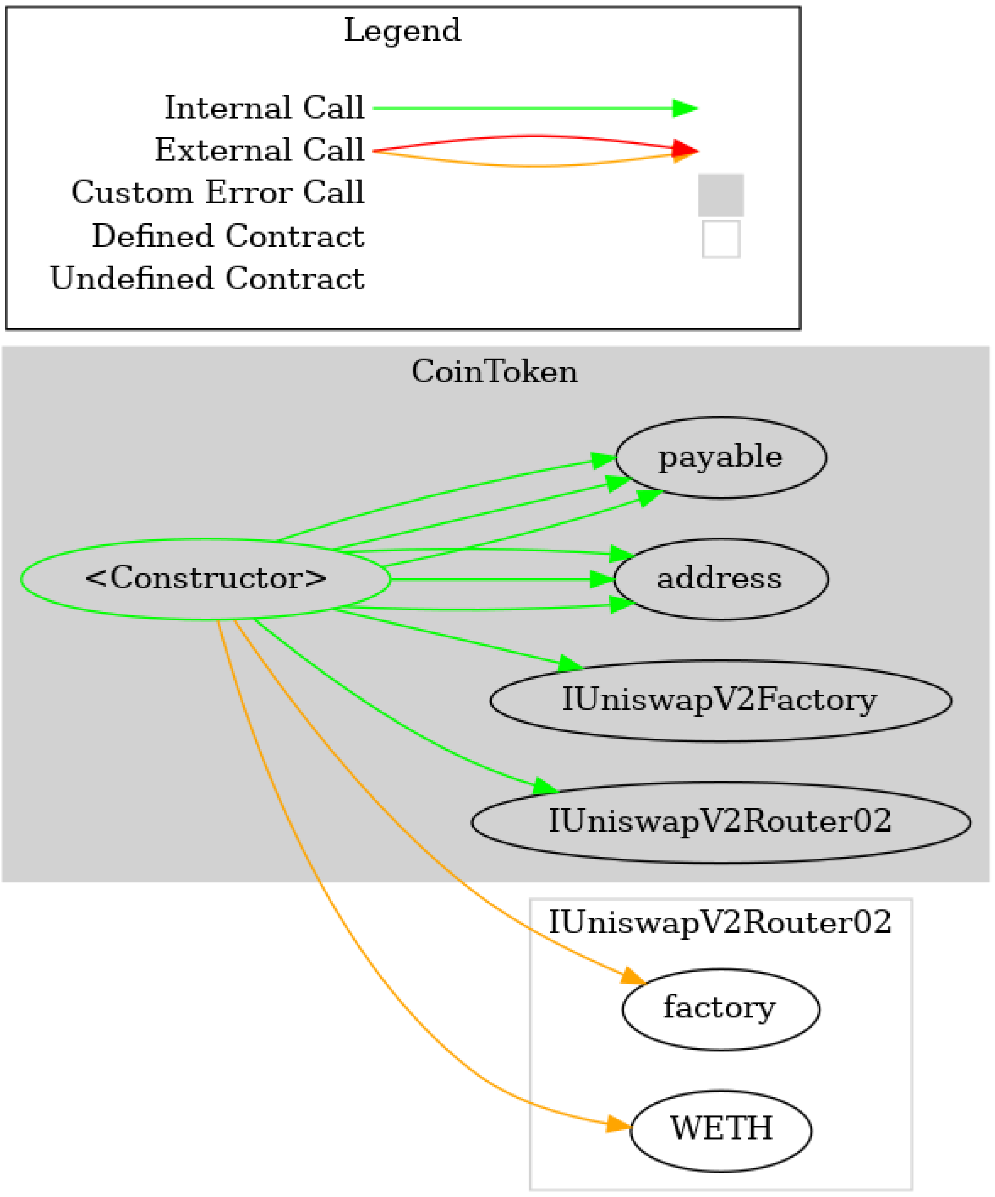
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CoinToken	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20 Ownable
		External	Payable	-
	mint	Private	✓	
	burn	Private	✓	
	_update	Internal	✓	
	addLqToUniswap	External	✓	onlyOwner
	openTrading	External	✓	onlyOwner
	addLiquidity	Private	✓	
	delayTransactionCheck	Private	✓	
	maxWalletCheck	Private		
	maxTransactionAmountCheck	Private		
	isFeeAppliedOnTransaction	Private		
	processFee	Private	✓	
	validateTotalFee	Private		
	shouldSwapBack	Private		
	swapAndLiquify	Private	✓	nonReentrant
	checkRatio	Private		

	swapTokensForEth	Private	✓	
	clearStuckToken	External	✓	onlyOwner
	manualSwap	External	✓	onlyOwner
	setFeeWallets	External	✓	onlyOwner
	setFees	External	✓	onlyOwner
	setFeeExclusionForAccount	External	✓	onlyOwner
	setExclusionFromMaxTransaction	External	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	addBots	External	✓	onlyOwner
	delBots	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	setSwapTokensAtAmountSupplyPercentage	External	✓	onlyOwner
	setSwapPossibility	External	✓	onlyOwner
	stopTransferDelay	External	✓	onlyOwner
	getTotalFeeAmount	Private		
	getBot	External		onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
CoinToken
<Constructor>
payable
address
IUniswapV2Factory
IUniswapV2Router02
IUniswapV2Router02
factory
WETH

Summary

KITTEN is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or two critical issues. The contract Owner can access some admin functions that can be used in a malicious way to disturb the users' transactions.

This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

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