Cyberscope

Audit Report

Kitten Token

December 2024

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

Commit : 492a56369617dc7059f3c103ca7e2f19261f8a8c

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Unresolved
	DDP	Decimal Division Precision	Unresolved
	PAMAR	Pair Address Max Amount Restriction	Unresolved
	OCTD	Transfers Contract's Tokens	Unresolved
	URR	Unhandled Router Reversion	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L16	Validate Variable Setters	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	492a56369617dc7059f3c103ca7e2f19261f8a8c
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	15 Nov 2024 https://github.com/cyberscope-io/audits/blob/main/4-kitten/v1/audit.pdf
Corrected Phase 2	06 Dec 2024

Source Files

Filename	SHA256
CoinToken.sol	fe82a8c44a94dd947e3aff17740fcf18f94d2d427448da19e06b161a2698ace2

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	CoinToken.sol
Status	Unresolved

Description

The contract owner has the authority to stop transactions, as described in detail in section `PAMAR`. As a result, the contract might operate as a honeypot.

Recommendation

The team is advised to follow the recommendations outlined in the `PAMAR` finding and implement the necessary steps to mitigate the identified risks, ensuring that the contract does not operate as a honeypot. Renouncing ownership will effectively eliminate the threats, but it is non-reversible.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	CoinToken.sol#L467,524
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 manualSwap() external onlyOwner returns (uint256, uint256,
uint256, uint256) {
        ...
    }

    function setFeeExclusionForAccount(address account, bool boolValue)
external onlyOwner {
       ...
    }

    function setExclusionFromMaxTransaction(address account, bool boolValue)
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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	CoinToken.sol#L301
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 feeAmount =
    (tokenAmount * (liquidityFee + devFee + marketingFee + charityFee) *
multiplier) / (100 * 1000);
uint256 burnAmount = (tokenAmount * burnFee * multiplier) / (100 * 1000);
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	CoinToken.sol#L259
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
    function maxWalletCheck(address to, uint256 amount) private view {
        if (to != pair && !isExcludedFromMaxTransactionAmount[to]) {
            if (amount + balanceOf(to) > maxWallet) {
                revert MaxWalletExceeded();
            }
        }
    }
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	CoinToken.sol#L459
Status	Unresolved

Description

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

```
   function clearStuckToken(address tokenAddress, uint256 tokens) external
onlyOwner returns (bool) {
        if (tokens == 0) {
            tokens = ERC20(tokenAddress).balanceOf(address(this));
        }

        return ERC20(tokenAddress).transfer(owner(), tokens);
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

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.

URR - Unhandled Router Reversion

Criticality	Minor / Informative
Location	CoinToken.sol#L227
Status	Unresolved

Description

The contract is designed to execute an external call to the router address to add liquidity via the `addLiquidityETH` function. However, if the liquidity addition fails for any reason (e.g., insufficient liquidity, invalid parameters, or router issues), the transaction will revert entirely. This behavior prevents the successful execution of the overall transaction and can lead to confusion, as the exact reason for failure may not be immediately clear or communicated to the user.

```
   function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private
returns (uint256, uint256, uint256) {
        if (balanceOf(address(this)) < tokenAmount) {
            revert InsufficientTokenBalance();
        }

        if (address(this).balance < ethAmount) {
            revert InsufficientEthBalance();
        }

        (uint256 amountTokenAddedToPool, uint256 amountETHAddedToPool,
uint256 amountLiquidityToken) =
            router.addLiquidityETH{value: ethAmount}(address(this),
tokenAmount, 0, 0, owner(), block.timestamp);

        return (amountTokenAddedToPool, amountETHAddedToPool,
amountLiquidityToken);
    }
```

Recommendation

It is recommended to execute the liquidity addition functionality within a try-catch block to handle potential reversion scenarios. By doing so, the contract can provide detailed feedback about the failure reason or implement alternative logic to proceed with partial functionality instead of reverting the entire transaction. This approach enhances the robustness and user experience of the contract.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	CoinToken.sol#L405,412
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bool success
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CoinToken.sol#L485,486,487
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(newDevWalletAddress)
marketingWallet = payable(newMarketingWalletAddress)
charityWallet = payable(newCharityWallet)
```

Recommendation

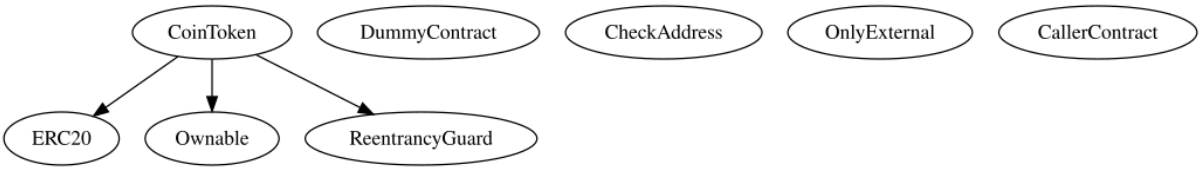
By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

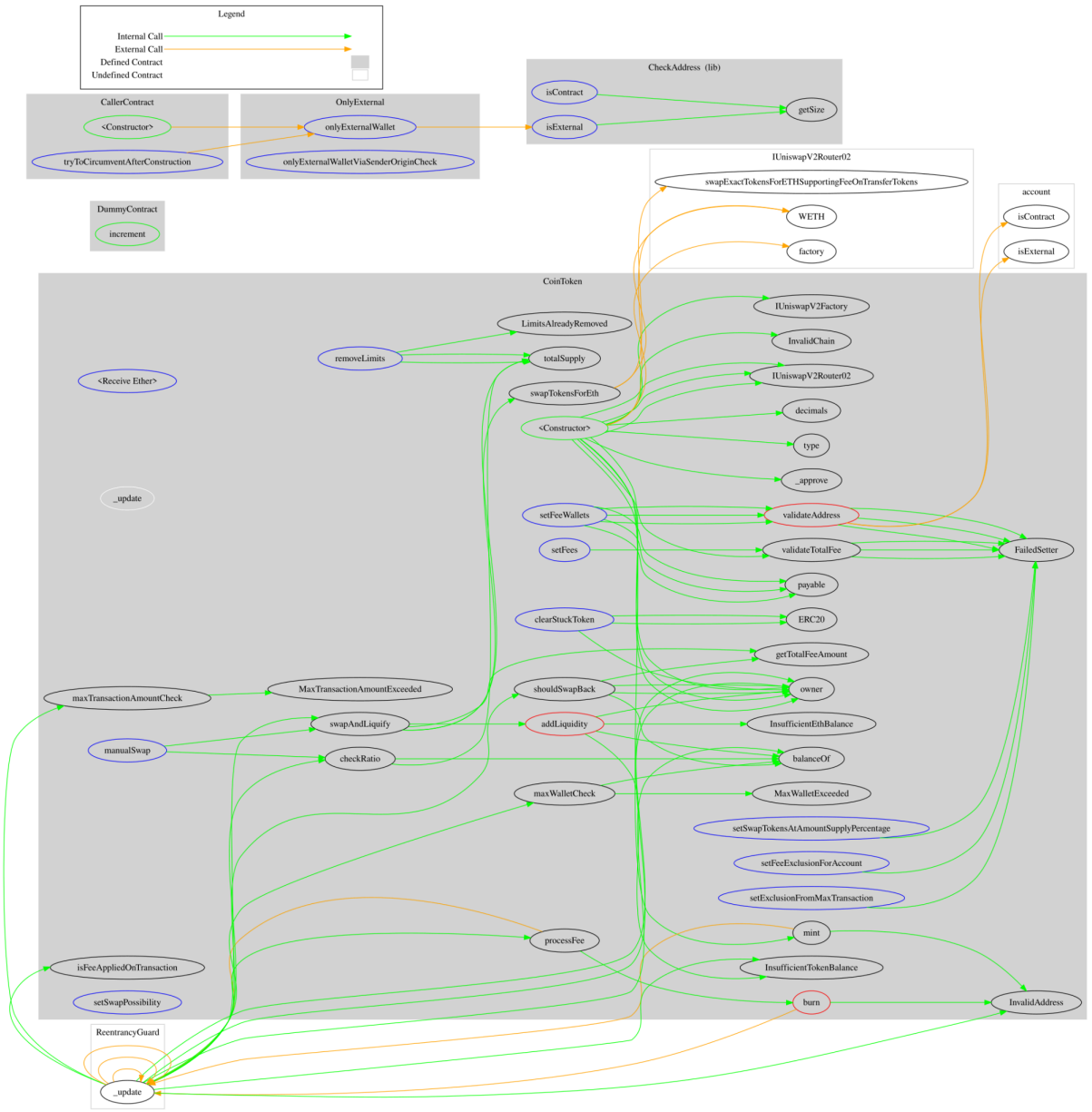
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CoinToken	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20 Ownable
		External	Payable	-
	mint	Private	✓	
	burn	Private	✓	
	_update	Internal	✓	
	addLiquidity	Private	✓	
	removeLimits	External	✓	onlyOwner
	maxWalletCheck	Private		
	maxTransactionAmountCheck	Private		
	isFeeAppliedOnTransaction	Private		
	processFee	Private	✓	
	validateTotalFee	Private		
	validateAddress	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
	setSwapTokensAtAmountSupplyPercentage	External	✓	onlyOwner
	setSwapPossibility	External	✓	onlyOwner
	getTotalFeeAmount	Private		

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
CallerContract
<Constructor>
tryToCircumventAfterConstruction
OnlyExternal
onlyExternalWallet
onlyExternalWalletViaSenderOriginCheck
CheckAddress (lib)
isContract
isExternal
getSize
IUniswapV2Router02
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
account
isContract
isExternal
DummyContract
increment
CoinToken
LimitsAlreadyRemoved
removeLimits
totalSupply
<Receive Ether>
swapTokensForEth
<Constructor>
IUniswapV2Factory
InvalidChain
IUniswapV2Router02
decimals
type
_approve
_update
setFeeWallets
setFees
validateAddress
validateTotalFee
FailedSetter
payable
clearStuckToken
ERC20
getTotalFeeAmount
maxTransactionAmountCheck
MaxTransactionAmountExceeded
shouldSwapBack
owner
swapAndLiquify
addLiquidity
InsufficientEthBalance
manualSwap
checkRatio
balanceOf
maxWalletCheck
MaxWalletExceeded
setSwapTokensAtAmountSupplyPercentage
setFeeExclusionForAccount
setExclusionFromMaxTransaction
processFee
mint
isFeeAppliedOnTransaction
InsufficientTokenBalance
setSwapPossibility
burn
InvalidAddress
ReentrancyGuard
_update

Summary

Kitten Token contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 5% 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

cyberscope.io