Cyberscope

Audit Report

Kraet.io

April 2024

Network	BASE
Address	0xa009De83199Be1489144c24501075cd0Da295fC4
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	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEFF	Missing Exclude Fees Functionality	Unresolved
●	UTA	Unnecessary Token Approval	Unresolved
●	ZTT	Zero Token Transfer	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L03	Redundant Statements	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved

Table of Contents

Review

Contract Name	KraetToken
Compiler Version	v0.8.25+commit.b61c2a91
Optimization	200 runs
Explorer	https://basescan.org/address/0xa009de83199be1489144c24501075cd0da295fc4
Address	0xa009de83199be1489144c24501075cd0da295fc4
Network	BASE
Symbol	Kraet
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Must Fix MEFF

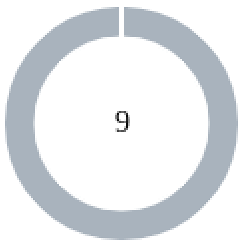
Audit Updates

Initial Audit	20 Apr 2024

Source Files

Filename	SHA256
kraetToken.sol	01c69ae7c87f81ab0a37aac37d15a0b89f2be8104b771166efcba8d1c609cbf8

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	9

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

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	kraetToken.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.

MEFF - Missing Exclude Fees Functionality

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

Description

The contract does not implement an exclude fees functionality, as a result it will not be able to operate in decentralized applications like lockers, presale launchapds etc.

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal override {
    uint256 taxAmount = 0;
    Flag flag = Flag.None;
    uint _fee = 5;
    if (to == swapPair) {
        taxAmount = amount * _fee / 100;
        flag = Flag.Sell;
    } else if (from == swapPair) {
        taxAmount = amount * _fee / 100;
        flag = Flag.Buy;
    }

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

Recommendation

The team is advised to implement an exclude from fees functionality so it will be able to operate correctly with decentralized applications.

UTA - Unnecessary Token Approval

Criticality	Minor / Informative
Location	kraetToken.sol#L559
Status	Unresolved

Description

It was identified that the contract's constructor grants an infinite approval to the Uniswap router for both the contract creator. This approval is granted through the `_approve` function with the maximum possible allowance value, `type(uint256).max`, effectvely allowing the Uniswap router to spend tokens from the contract creator without limit.

```
constructor() ERC20("Kraet", "Kraet") {
    uint256 startSupply = 0.1e10 * 10 ** decimals();
    _mint(msg.sender, (startSupply));

    IUniswapV2Router _uniswapRouter = IUniswapV2Router(
        0x4752ba5DBc23f44D87826276BF6Fd6b1C372aD24
    );
    swapPair = IUniswapV2Factory(_uniswapRouter.factory())
        .createPair(address(this), _uniswapRouter.WETH());

    swapRouter = _uniswapRouter;

    _approve(msg.sender, address(swapRouter), type(uint256).max);
    _approve(address(this), address(swapRouter), type(uint256).max);
}
```

Recommendation

It is advised to review the necessity of granting infinite approval to the Uniswap router during contract deployment. If this approval is not essential for the contract's intended functionality, it should be removed or limited to the necessary allowance amount required for specific operations.

By restricting approval to the minimum necessary level, the contract can minimize the potential impact of malicious activities, such as unauthorized token transfers or manipulative trading behaviors, while still maintaining functionality with external protocols like Uniswap.

ZTT - Zero Token Transfer

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

Description

The `_transfer` function within the contract is designed to facilitate transfers between addresses, applying a tax only when the transfer involves buying or selling tokens. However, it neglects to consider scenarios where the transfer does not qualify as a buy or sell. When the transfer is not a buy or sell, it will still transfer zero tokens to the `_taxWallet`.

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal override {
    uint256 taxAmount = 0;
    Flag flag = Flag.None;
    uint _fee = 5;
    if (to == swapPair) {
        taxAmount = amount * _fee / 100;
        flag = Flag.Sell;
    } else if (from == swapPair) {
        taxAmount = amount * _fee / 100;
        flag = Flag.Buy;
    }

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

Recommendation

To ensure precise tax handling within the contract, it's recommended to implement a validation step before transferring the tax amount to the `_taxWallet`. This step should verify whether the calculated `taxAmount` is greater than zero. If the `taxAmount` exceeds zero, proceed with the transfer to the `_taxWallet` otherwise, bypass the transfer.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	kraetToken.sol#L517,518,524,532,533,534
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 = 0xa03f74644a7230521E961Ab4b22c32b82175A5e3
address public _ownerWallet = 0xd01762283448FDC0f07Aae6f21a3394c33db6625
uint256 private _cnt_tx = 0
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.

L03 - Redundant Statements

Criticality	Minor / Informative
Location	kraetToken.sol#L583,597
Status	Unresolved

Description

Redundant statements are statements that are unnecessary or have no effect on the contract's behavior. These can include declarations of variables or functions that are not used, or assignments to variables that are never used.

As a result, it can make the contract's code harder to read and maintain, and can also increase the contract's size and gas consumption, potentially making it more expensive to deploy and execute.

```
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

To avoid redundant statements, it's important to carefully review the contract's code and remove any statements that are unnecessary or not used. This can help to improve the clarity and efficiency of the contract's code.

By removing unnecessary or redundant statements from the contract's code, the clarity and efficiency of the contract will be improved. Additionally, the size and gas consumption will be reduced.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	kraetToken.sol#L98,517,518,521,524,528
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 = 0xa03f74644a7230521E961Ab4b22c32b82175A5e3
address public _ownerWallet = 0xd01762283448FDC0f07Aae6f21a3394c33db6625
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	kraetToken.sol#L521,524,528,532,533,534
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 _fisrt_block_num = block.number
uint256 private _cnt_tx = 0
uint256 public _maxTokens = 100 * 10 ** decimals();
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	kraetToken.sol#L425,583
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);
}
...
function _addLiquidity(
    uint256 tokenAmount,
    uint256 ethAmount
) private {
    swapRouter.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0,
        0,
        owner(),
        block.timestamp
    );
}
```

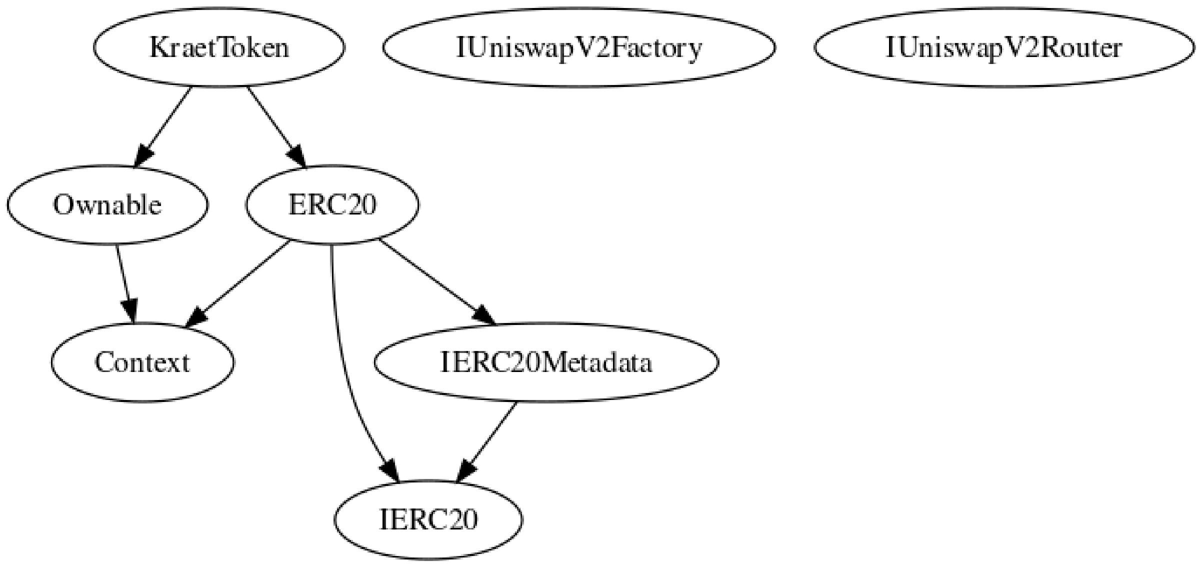
Recommendation

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.

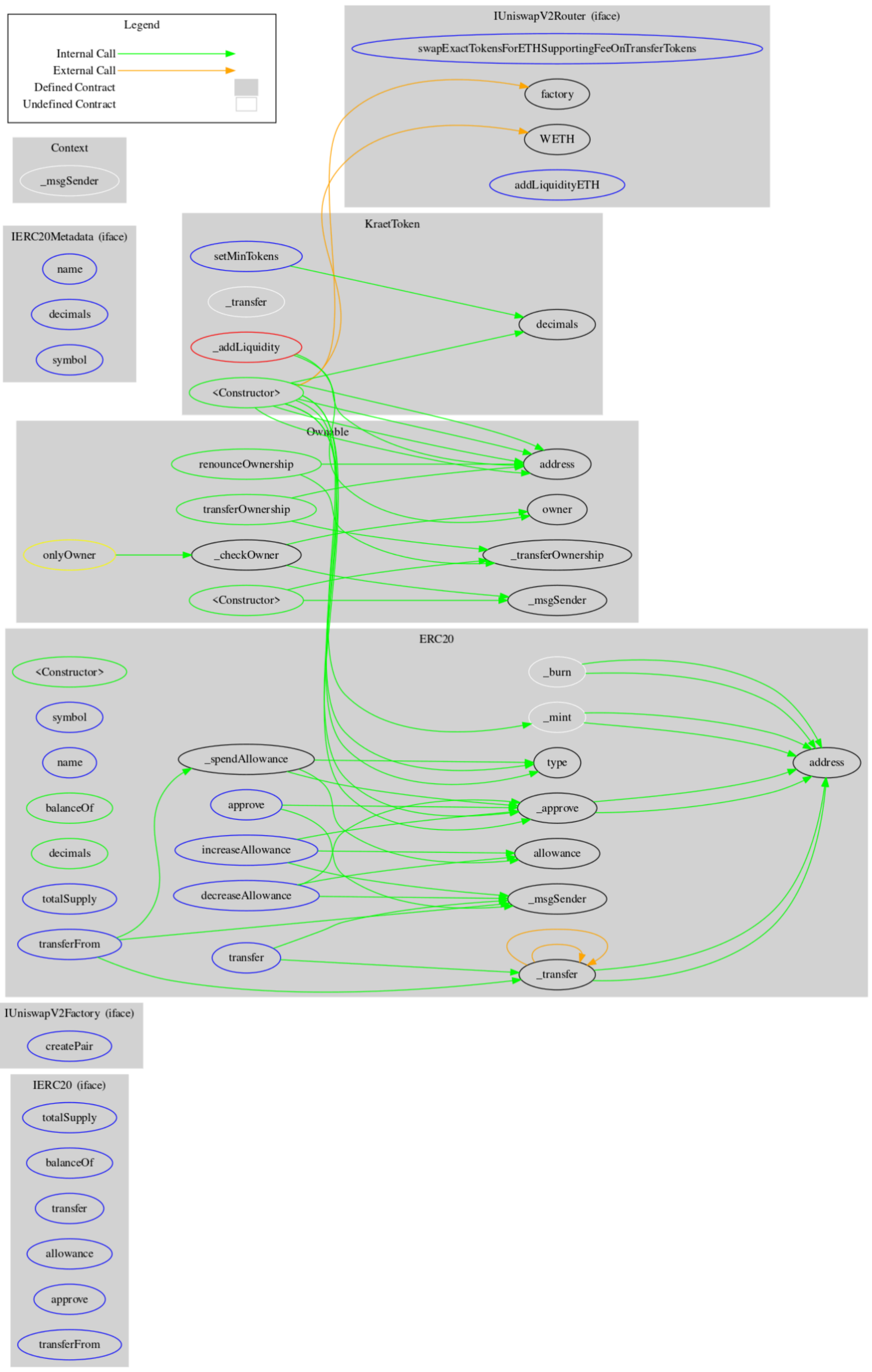
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
KraetToken	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
IUniswapV2Router (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
factory
WETH
addLiquidityETH
KraetToken
setMinTokens
_transfer
_addLiquidity
<Constructor>
decimals
Ownable
renounceOwnership
transferOwnership
onlyOwner
_checkOwner
<Constructor>
address
owner
_transferOwnership
_msgSender
ERC20
<Constructor>
symbol
name
balanceOf
decimals
totalSupply
transferFrom
_spendAllowance
approve
increaseAllowance
decreaseAllowance
transfer
_burn
_mint
type
_approve
allowance
_msgSender
_transfer
address
IUniswapV2Factory (iface)
createPair
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Kraet.io contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error or critical issues. There is also a fee of 5% on buys and sells.

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