Cyberscope

Audit Report

Orenium protocol

March 2024

Network	BSC
Address	0xb0Dba141b38E61d704168faB3Ce7366575C503ad
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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	EPC	Existing Pair Creation	Unresolved
	FRV	Fee Restoration Vulnerability	Unresolved
	MEE	Missing Events Emission	Unresolved
	MTEE	Missing Transfer Event Emission	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PMRM	Potential Mocked Router Manipulation	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RED	Redundant Event Declaration	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L05	Unused State Variable	Unresolved

●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	OreniumCoin
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0xb0dba141b38e61d704168fab3ce7366575c503ad
Address	0xb0dba141b38e61d704168fab3ce7366575c503ad
Network	BSC
Symbol	ORE
Decimals	18
Total Supply	125,000,000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	13 Mar 2024

Source Files

Filename	SHA256
OreniumCoin.sol	a802bcbe836681f0b603b938a9922d4e8e2cf7a2649ca1bb63a484c95972f31d

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	20

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	20	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	OreniumCoin.sol#L592,595
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxOreHold` or `_maxTrx_ore` to zero. As a result, the contract may operate as a honeypot.

```
if (to != owner() &&
    to != Admin_Wallet &&
    to != address(this) &&
    to != uniswapV2Pair &&

    from != owner()){
    uint256 heldTokens = balanceOf(to);
    require((heldTokens + amount) <= _maxOreHold,"You are
trying to buy too many tokens. You have reached the limit for
one wallet.");}

if (from != owner() && to != owner())
    require(amount <= _maxTrx_ore, "You are trying to buy more
than the max transaction limit.");
```

Additionally, the contract owner has the authority to stop transactions, as described in detail in sections PTRP, PVC, PMRM and PLPI. As a result, the contract might operate as a honeypot.

Recommendation

The contract could embody a check for not allowing setting the `_maxOreHold` and the `_maxTrx_ore` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	OreniumCoin.sol#L508
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `_ore_fee_settings` function with new buy and sell fees that have to add up to 30%.

```
uint256 private maxPossibleFee = 30;

function _ore_fee_settings(uint256 Ore_buy_update, uint256
Ore_sell_update) external onlyOwner() {

    require((Ore_buy_update + Ore_sell_update) <=
maxPossibleFee, "Fee is too high!");
    Ore_sell_fee = Ore_sell_update;
    Ore_buy_fee = Ore_buy_update;

}
```

Recommendation

The contract could embody a check for the maximum acceptable value and not surpass the total limit of 25%. 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.

EPC - Existing Pair Creation

Criticality	Minor / Informative
Location	OreniumCoin.sol#L666
Status	Unresolved

Description

The contract contains a function that does not handle the scenario where a pair already exists prior to its execution. If a pair for the given tokens has already been established, the `createPair` function will revert and not proceed with the creation of a new pair. As a result, if a pair has been previously set up before the function is invoked, the contract will encounter an error when trying to call the `createPair` function. This will prevent the successful execution, essentially leading the function to revert.

```
function New_Router_and_Update_Pair(address newRouter) public
onlyOwner() {
    IUniswapV2Router02 _newPCSRouter =
IUniswapV2Router02(newRouter);
    uniswapV2Pair =
IUniswapV2Factory(_newPCSRouter.factory()).createPair(address(t
his), _newPCSRouter.WETH());
    uniswapV2Router = _newPCSRouter;
}
```

Recommendation

To mitigate the risks associated with attempting to create an already existing pair, it is recommended to implement a check to determine whether the pair already exists before proceeding to create a new pair. This can be achieved by utilizing the getPair function of the Factory contract to retrieve the address of the pair contract for the specified tokens. If the address returned by the getPair function is the zero address, it indicates that the pair does not exist, and the contract can proceed with the createPair function. Conversely, if a non-zero address is returned, it indicates that the pair already exists, and the createPair function will revert.

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	OreniumCoin.sol#L543,549,682
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
function removeAllFee() private {
    if(_TotalFee == 0 && Ore_buy_fee == 0 && Ore_sell_fee == 0)
return;


    _previousBuyFee = Ore_buy_fee;
    _previousSellFee = Ore_sell_fee;
    _previousTotalFee = _TotalFee;
    Ore_buy_fee = 0;
    Ore_sell_fee = 0;
    _TotalFee = 0;

}

function restoreAllFee() private {

_TotalFee = _previousTotalFee;
Ore_buy_fee = _previousBuyFee;
Ore_sell_fee = _previousSellFee;

}

function _tokenTransfer(address sender, address recipient,
uint256 amount,bool takeFee) private {


    if(!takeFee){
        removeAllFee();
        } else {
            txCount++;
        }
        _transferTokens(sender, recipient, amount);

    if(!takeFee)
        restoreAllFee();
}
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	OreniumCoin.sol#L508,526,535,539,544
Status	Unresolved

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.

```
function _ore_fee_settings(uint256 Ore_buy_update, uint256
Ore_sell_update) external onlyOwner() {

    require((Ore_buy_update + Ore_sell_update) <=
maxPossibleFee, "Fee is too high!");
    Ore_sell_fee = Ore_sell_update;
    Ore_buy_fee = Ore_buy_update;

}

function Transfers_tax_Update(bool true_or_false) external
onlyOwner {
    checkfeetransfer_ = true_or_false;
}

...
```

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.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	OreniumCoin.sol#L700
Status	Unresolved

Description

The contract is a missing transfer event emission when fees are transferred to the contract address as part of the transfer process. This omission can lead to a lack of visibility into fee transactions and hinder the ability of decentralized applications (DApps) like blockchain explorers to accurately track and analyze these transactions.

```
_tOwned[address(this)] = _tOwned[address(this)].add(tDev);
```

Recommendation

To address this issue, it is recommended to emit a transfer event after transferring the taxed amount to the contract address. The event should include relevant information such as the sender, recipient, and the amount transferred.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	OreniumCoin.sol#L649
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 swapTokensForBNB(uint256 tokenAmount) private {

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router),
tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
        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.

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	OreniumCoin.sol#L666,672
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. 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.

```
function New_Router_and_Update_Pair(address newRouter) public
onlyOwner() {
    IUniswapV2Router02 _newPCSRouter =
IUniswapV2Router02(newRouter);
    uniswapV2Pair =
IUniswapV2Factory(_newPCSRouter.factory()).createPair(address(t
his), _newPCSRouter.WETH());
    uniswapV2Router = _newPCSRouter;
}

function New_Router_Address(address newRouter) public
onlyOwner() {
    IUniswapV2Router02 _newPCSRouter =
IUniswapV2Router02(newRouter);
    uniswapV2Router = _newPCSRouter;
}
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	OreniumCoin.sol#L626
Status	Unresolved

Description

The contract sends funds to an `Admin_Wallet` 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.

```
function sendToWallet(address payable wallet, uint256 amount)
private {
        wallet.transfer(amount);
    }
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	OreniumCoin.sol#L526,609
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for BNB. The variable `swapTrigger` sets the number of transactions, after which the swap functionality will be triggered. If the variable is set to a big number, then the contract will swap a huge amount of tokens for BNB, since `_maxTrx_ore` can also be set to a big number.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
if(
    txCount >= swapTrigger &&
    !inSwapAndLiquify &&
    from != uniswapV2Pair &&
    swapAndLiquifyEnabled
    )
{

    txCount = 0;
    uint256 contractTokenBalance = balanceOf(address(this));
    if(contractTokenBalance > _maxTrx_ore)
{contractTokenBalance = _maxTrx_ore;}
    if(contractTokenBalance > 0){
    swapAndLiquify(contractTokenBalance);
}
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	OreniumCoin.sol#L418
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 contract declares certain events in its code. However, these events are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiqudity

);
```

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.

A recommended approach would be to either remove the declared events that are not being emitted or to incorporate the necessary emit statements within the contract's functions to actually emit these events when relevant actions occur.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	OreniumCoin.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	OreniumCoin.sol#L499,503,516,526,535,539,543
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function excludeTaxLimit(address account) public onlyOwner {
    checknofee_transfer[account] = true;
}

function includeInTaxLimit(address account) public onlyOwner {
    checknofee_transfer[account] = false;
}

...
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	OreniumCoin.sol#L385,386,387,388,389,394
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.

```
string private _name = "Orenium Protocol"
string private _symbol = "ORE"
uint8 private _decimals = 18
uint256 private _tTotal = 125000000 * 10**18
uint256 private _tFeeTotal
uint256 private maxPossibleFee = 30
```

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	OreniumCoin.sol#L210,211,224,241,380,383,400,401,402,404,408,508,516,521,526,535,539,541,639,666,672,677
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
mapping (address => bool) public checknofee_transfer
address payable public Admin_Wallet =
payable(0xACcF90d8714b4795F43b6a505E6031a6dC02A75C)
uint256 private _TotalFee = 30
uint256 public Ore_buy_fee = 15
uint256 public Ore_sell_fee = 15
uint256 public _maxOreHold = _tTotal.mul(2).div(100)
uint256 public _maxTrx_ore = _tTotal.mul(2).div(100)
uint256 Ore_buy_update

...
```

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	OreniumCoin.sol#L389,405,409
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 _tFeeTotal
uint256 private _previousMaxWalletToken = _maxOreHold
uint256 private _previousMaxTxAmount = _maxTrx_ore
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	OreniumCoin.sol#L511,527,540,544
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.

```
Ore_sell_fee = Ore_sell_update
swapTrigger = number_of_transactions
_maxOreHold = _tTotal*maxWallPercent_x100/10000
_maxTrx_ore = _tTotal*maxTxPercent_x100/10000
```

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	OreniumCoin.sol#L87,93,99,103,107,111,118,122,129,133,139
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 isContract(address account) internal view returns
(bool) {
        uint256 size;
        assembly { size := extcodesize(account) }
        return size > 0;
    }

...
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level
call failed");
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	OreniumCoin.sol#L522,678
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.

```
Admin_Wallet = wallet
uniswapV2Pair = newPair
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	OreniumCoin.sol#L89,144
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	OreniumCoin.sol#L24
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 ^0.8.7;
```

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	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	sub	Internal		
	div	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-

	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-

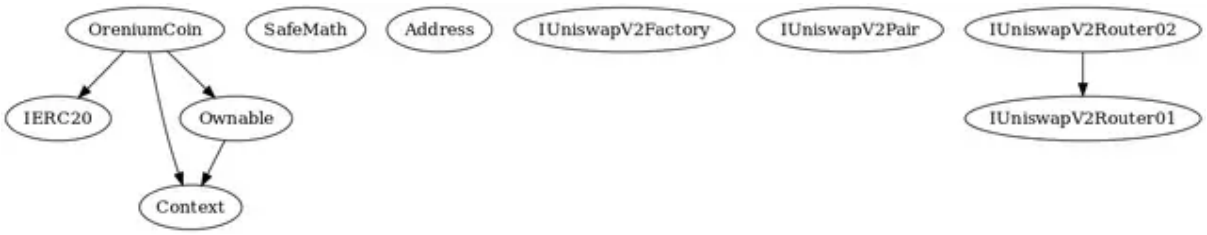
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-

	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
OreniumCoin	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-

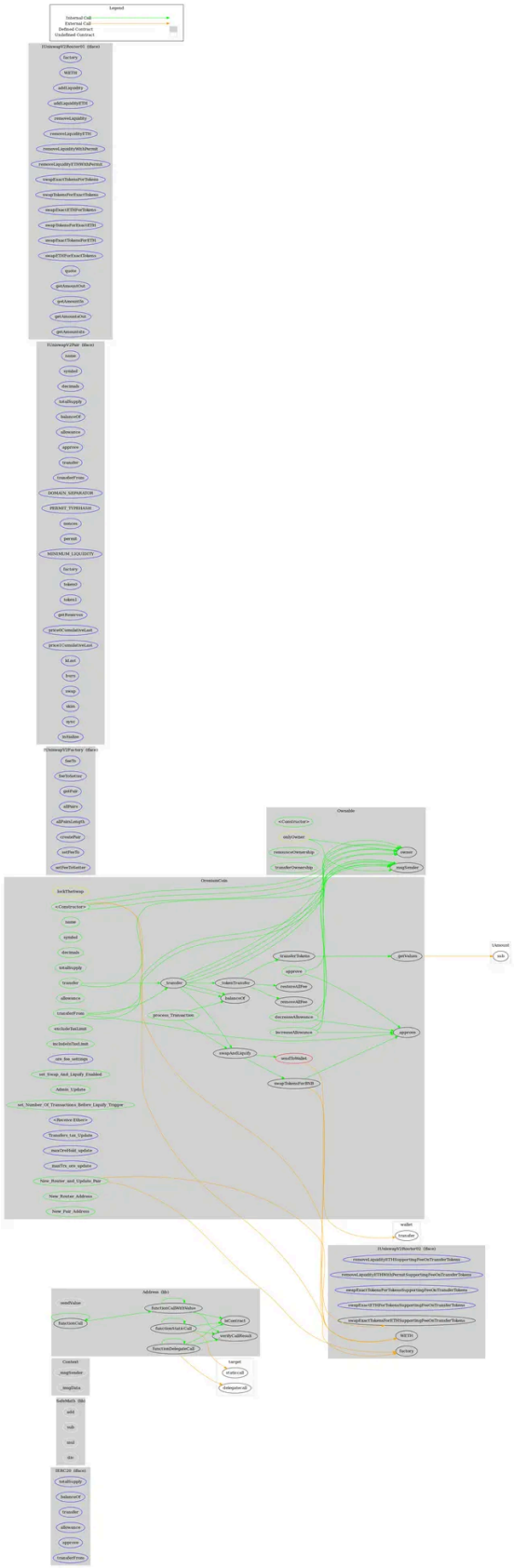
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	excludeTaxLimit	Public	✓	onlyOwner
	includeInTaxLimit	Public	✓	onlyOwner
	_ore_fee_settings	External	✓	onlyOwner
	set_Swap_And_Liquify_Enabled	Public	✓	onlyOwner
	Admin_Update	Public	✓	onlyOwner
	set_Number_Of_Transactions_Before_Liquify_Trigger	Public	✓	onlyOwner
		External	Payable	-
	Transfers_tax_Update	External	✓	onlyOwner
	_maxOreHold_update	External	✓	onlyOwner
	_maxTrx_ore_update	External	✓	onlyOwner
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	_approve	Private	✓	
	_transfer	Private	✓	

	sendToWallet	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	process_Transaction	Public	✓	onlyOwner
	swapTokensForBNB	Private	✓	
	New_Router_and_Update_Pair	Public	✓	onlyOwner
	New_Router_Address	Public	✓	onlyOwner
	New_Pair_Address	Public	✓	onlyOwner
	_tokenTransfer	Private	✓	
	_transferTokens	Private	✓	
	_getValues	Private		

Inheritance Graph



Flow Graph



Summary

Orenium protocol 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 and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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