Cyberscope

# Audit Report

# THORIUMFI

February 2024

| | |
|---|---|
| Network | BSC |
| Address | 0xf6dbcf01f28bc43f35fe23a89108c1e090304478 |
| 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 |
|---|---|---|---|
| | DDP | Decimal Division Precision | Unresolved |
| | MEE | Missing Events Emission | Unresolved |
| | PLPI | Potential Liquidity Provision Inadequacy | Unresolved |
| | RES | Redundant Event Statement | Unresolved |
| | RSD | Redundant Swap Duplication | Unresolved |
| | L02 | State Variables could be Declared Constant | Unresolved |
| | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| | L09 | Dead Code Elimination | Unresolved |
| | L16 | Validate Variable Setters | Unresolved |
| | L19 | Stable Compiler Version | Unresolved |
| | L20 | Succeeded Transfer Check | Unresolved |

# Table of Contents

# Review

| | |
|---|---|
| **Contract Name** | Thorium |
| **Compiler Version** | v0.8.12+commit.f00d7308 |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0xf6dbcf01f28bc43f35fe23a89108c1e090304478 |
| **Address** | 0xf6dbcf01f28bc43f35fe23a89108c1e090304478 |
| **Network** | BSC |
| **Symbol** | THOR |
| **Decimals** | 18 |
| **Total Supply** | 12,000,000 |
| **Badge Eligibility** | Yes |

## Audit Updates

| | |
|---|---|
| **Initial Audit** | 21 Feb 2024 |

## Source Files

| | |
|---|---|
| **Filename** | SHA256 |
| **Thorium.sol** | c5ba063404ce5982ca4160d9c2ff019e78a2e506baa1b5b47e9792698bc5a1be |

# Findings Breakdown


| | |
|---|---|
| Critical | 0 |
| Medium | 0 |
| Minor / Informative | 11 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 |
| Minor / Informative | 11 | 0 | 0 | 0 |

# DDP - Decimal Division Precision

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L598 |
| **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.

```
if(liquidityShare > 0) {
    uint256 liquidityTokens = contractTokenBalance *
liquidityShare / 100;
    swapAndLiquify(liquidityTokens);
}

if(marketingShare > 0) {
    uint256 marketingTokens = contractTokenBalance *
marketingShare / 100;
    swapAndSendMarketing(marketingTokens);
}
```

## 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.

# MEE - Missing Events Emission

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L555,561,699 |
| **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 updateMarketingWalletShares(uint256 _mkShareM, uint256
_mkShareS) external onlyOwner {
    require(_mkShareM + _mkShareS == 100, "Marketing fee shares
must add up to 100");
    mkShareM = _mkShareM;
    mkShareS = _mkShareS;
    }

function enableWalletToWalletTransferWithoutFee(bool enable)
external onlyOwner {
    require(walletToWalletTransferWithoutFee != enable, "Wallet
to wallet transfer without fee is already set to that value");
    walletToWalletTransferWithoutFee = enable;
}
```

## 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.

## PLPI - Potential Liquidity Provision Inadequacy

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L645,673 |
| **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.

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

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
    half,
    0, // accept any amount of ETH
    path,
    address(this),
    block.timestamp);
```

## Recommendation

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

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

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

## RES - Redundant Event Statement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L458 |
| **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 `FeesUpdated` event statement is not used in the contract's implementation.

```
event FeesUpdated(uint256 buyFee, uint256 sellFee);
```

### 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. It recommended to remove the unused event statement from the contract.

# RSD - Redundant Swap Duplication

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L598 |
| **Status** | Unresolved |

## Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
if(liquidityShare > 0) {
    uint256 liquidityTokens = contractTokenBalance *
liquidityShare / 100;
    swapAndLiquify(liquidityTokens);
}

if(marketingShare > 0) {
    uint256 marketingTokens = contractTokenBalance *
marketingShare / 100;
    swapAndSendMarketing(marketingTokens);
}
```

## Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

## L02 - State Variables could be Declared Constant

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L432,433,449 |
| **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.

```
uint256 public buyFee = 2
uint256 public sellFee = 2
address private DEAD =
0x000000000000000000000000000000000000dEaD
```

### 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** | Thorium.sol#L260,261,278,298,449,548,555,566 |
| **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);
address private DEAD =
0x000000000000000000000000000000000000dEaD
uint256 _liquidityFeeShare
uint256 _marketingFeeShare
uint256 _mkShareM
uint256 _mkShareS
address _mkWalletS
address _mkWalletM
```

## 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.

## L09 - Dead Code Elimination

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L190 |
| **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");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
        }
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }
```

## 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** | Thorium.sol#L520 |
| **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.

```
uniswapV2Pair = _uniswapV2Pair
```

### 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** | Thorium.sol#L7 |
| **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 ^0.8.0;
```

### 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.

## L20 - Succeeded Transfer Check

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Thorium.sol#L500 |
| **Status** | Unresolved |

### Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
ERC20token.transfer(msg.sender, balance)
```

### Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **IERC20Metadata** | Interface | IERC20 | | |
| | name | External | | - |
| | symbol | External | | - |
| | decimals | External | | - |
| | | | | |
| **Context** | Implementation | | | |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | | | | |
| **Ownable** | Implementation | Context | | |
| | | Public | ✓ | - |

| | | | | |
|---|---|---|---|---|
| | owner | Public | | - |
| | renounceOwnership | Public | ✓ | onlyOwner |
| | transferOwnership | Public | ✓ | onlyOwner |
| | | | | |
| **ERC20** | Implementation | Context, IERC20, IERC20Metadata | | |
| | | Public | ✓ | - |
| | name | Public | | - |
| | symbol | Public | | - |
| | decimals | Public | | - |
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | allowance | Public | | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | _transfer | Internal | ✓ | |
| | _mint | Internal | ✓ | |
| | _burn | Internal | ✓ | |
| | _approve | Internal | ✓ | |
| | _beforeTokenTransfer | Internal | ✓ | |

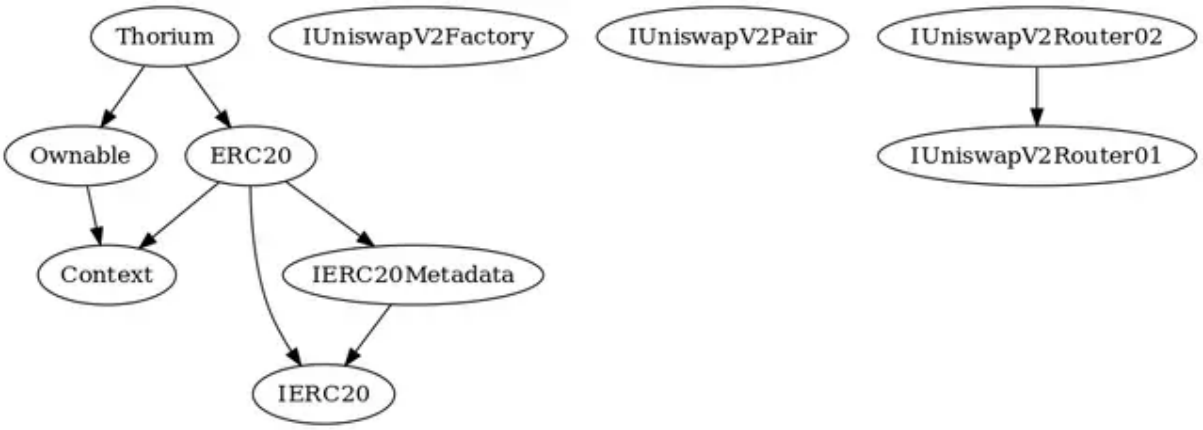
| | | | | |
|---|---|---|---|---|
| | _afterTokenTransfer | Internal | ✓ | |
| | | | | |
| **IUniswapV2Factory** | Interface | | | |
| | feeTo | External | | - |
| | feeToSetter | External | | - |
| | getPair | External | | - |
| | allPairs | External | | - |
| | allPairsLength | External | | - |
| | 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 | | - |
| | mint | 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 | ✓ | - |

| | | | | |
|---|---|---|---|---|
| **Thorium** | Implementation | ERC20, Ownable | | |
| | | Public | ✓ | ERC20 |
| | | External | Payable | - |
| | claimStuckTokens | External | ✓ | onlyOwner |
| | isContract | Internal | | |
| | sendBNB | Internal | ✓ | |
| | updateUniswapV2Router | External | ✓ | onlyOwner |
| | setAutomatedMarketMakerPair | External | ✓ | onlyOwner |
| | _setAutomatedMarketMakerPair | Private | ✓ | |
| | excludeFromFees | External | ✓ | onlyOwner |
| | isExcludedFromFees | Public | | - |
| | updateFeeShares | External | ✓ | onlyOwner |
| | updateMarketingWalletShares | External | ✓ | onlyOwner |
| | enableWalletToWalletTransferWithoutFee | External | ✓ | onlyOwner |
| | changeMarketingWallet | External | ✓ | onlyOwner |
| | _transfer | Internal | ✓ | |
| | swapAndLiquify | Private | ✓ | |
| | swapAndSendMarketing | Private | ✓ | |
| | setSwapTokensAtAmount | External | ✓ | onlyOwner |

# Inheritance Graph



# Flow Graph

# Summary

THORIUMFI contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. THORIUMFI is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract ownership has been renounced.

# 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